Universidad Carlos III de Madrid

Escuela Politécnica Superior

Departamento: Mecánica de Medios Continuos y Teoría de Estructuras

Tutores ENSAM de París: Michel Vergé

Tutor UC3M: Dr. Ángel Arias Hernández

INGENIERÍA INDUSTRIAL

PROYECTO FIN DE CARRERA

OPTIMIZACIÓN DE AUTÓMATAS ORIENTADOS A ASISTENCIA DE PERSONAS CON MOVILIDAD REDUCIDA

Khalid Bashir Rodríguez

Septiembre, 2010

ANNEE : 2009-2010

Groupe et numéro du PJE : PA-F10073

CENTRE DE RATTACHEMENT PJE : Paris

AUTEUR : BASHIR RODRÍGUEZ Khalid

TITRE : *Optimización de autómatas orientados a asistencia de personas con movilidad reducida*

ENCADREMENT DU PJE : Ángel Arias; Michel Vergé ;

PARTENAIRE DU PJE : HanditecAM

NOMBRE DE PAGES : 89

NOMBRE DE REFERENCES BIBLIOGRAPHIQUES : 8

RESUME : Ce travail est fait sur le robot HAMMI qui est un robot destiné a l´assistance de personnes handicapées pour pousser n´importe quel fauteuil roulant manuel. Après plusieurs études menées les années précédentes, l´objectif de ce projet est de faire évoluer le prototype existant en tenant compte des informations fournies par les capteurs pour définir les trajectoires possibles et éviter les obstacles. La réalisation de cet objectif est faite avec la programmation dans l'environnement Visual C++ de Microsoft et avec les bibliothèques des cartes National Instrument.

MOTS-CLES : Robotique mobile, personnes handicapées, Visual C++, Trajectographie.

REMERCIEMENTS

Je voudrais commencer par remercier toute ma famille pour son soutien inconditionnel. Dans le lointain, elle a toujours été près de moi, de mes bonheurs et de mes malheurs. Sans son effort, je n'aurais jamais réussi dans mes études.

Je remercie également M. Vergé pour son encadrement dans ce projet. Il m´a aidé beaucoup et cela m´a permis de développer mon projet dans une ambiance très propice.

Je ne pourrai jamais oublier Mme. Cliton qui a été toujours disponible et qui a dédié tout son effort et son temps à m'offrir un enseignement du français le plus complet possible.

Finalement, je remercie tous les amis que j´ai connu ici, ils m'apportent des bons moments et leur soutien et aide tout le temps.

Synthèse en espagnol

1. ÍNDICE

1. ÍNDICE 6
2. INTRODUCCIÓN 7
a) El robot HAMMI 7
- Parte mecánica y eléctrica. 7
- Parte electrónica. 8

3. PROGRAMACION DEL ROBOT. Visual C++ y National Instruments 10
a) Arquitectura Document-View, Visual Studio 2005 y NI. 10
4. PROGRAMACION DE LA TRAYECTORIA PARA EVITAR OBSTACULOS 12
a) Programa existente. Robot_HAMMI v2.2 12
b) Desarrollo del nuevo programa. Robot_HAMMI v3.2 14
- Metodología seguida para evitar los obstáculos 15
- Modificación del código para evitar obstáculos 19
- Modificación del código para guardar los datos 20

c) Creación de un programa para el análisis de los datos en MATLAB 21
d) Validación del método empleado 22
5. CONCLUSIONES Y TRABAJOS FUTUROS 28
6. BIBLIOGRAFIA 29

2. INTRODUCCIÓN

Desde siempre, la tecnología ha estado enfocada al servicio de la humanidad y está destinada a satisfacer sus necesidades. Por lo tanto, es de esperar que a medida que la tecnología evolucione, será puesta a disposición de las personas discapacitadas. Este es el fin de este proyecto.

En el año 2007, la población francesa con una deficiencia motriz sobrepasaba el 10%, llegando al 21% si se consideraban otras deficiencias. En el caso de la comunidad europea, los datos son parecidos. Queda el presente proyecto inscrito en este marco. Si se quiere profundizar más en estos datos se puede consultar la referencia bibliográfica [1].

De tal forma, la ENSAM interesada en hacer una contribución a este campo, propuso el proyecto llamado Robótica Móvil Para Personas Discapacitadas. Siempre con la colaboración de la sociedad HandiTecAM, el fin de este proyecto es el de diseñar una plataforma de desarrollo que permita ayudar a las personas con deficiencias motrices o sensoriales.

Dentro de este gran proyecto desde hace tres años, se desarrolla el robot HAMMI (HandiTec Arts et Métiers Motorización Inteligente). Se trata de un prototipo de robot autónomo que, acoplado detrás de una silla de ruedas, será capaz de seguir las instrucciones del ocupante o desplazarse automáticamente.

a) El robot HAMMI

En este epígrafe se presentará el estado del robot HAMMI al comienzo de este proyecto. Dicha presentación se estructurará según dos partes: primero los componentes mecánicos y eléctricos y seguidamente, los componentes electrónicos.

La primera parte es necesaria para tener una concepción global sobre el robot. La segunda parte es la más relacionada con el presente proyecto y por lo tanto, es imprescindible para entender el trabajo realizado.

Parte mecánica y eléctrica.

Gracias a los proyectos realizados en los últimos años por varios equipos de trabajo sobre el robot HAMMI, se dispone de una estructura autónoma móvil capaz de unir y portar los componentes necesarios para el funcionamiento adecuado del sistema.

Los elementos más importantes y sus funciones se exponen en la siguiente tabla:

COMPONENTE	DESCRIPCIÓN	FUNCIÓN
Estructura de soporte	Conjunto de segmentos de perfil estructural	**Sirve de soporte para los otros componentes y asegura la estabilidad**
Ruedas motrices	Ruedas laterales con motores eléctricos y encoders	**Desplazamiento del robot y control de las trayectorias del mismo**
Fuente de alimentación	Alimentación regulada 24 v.	**Alimentar toda la electrónica excepto el ordenador embarcado**
Ventosas magnéticas	Electroimanes cilíndricos	**Transmisión de fuerzas a la silla**
Cintura para sensores	Soporte metálico mecanizado	**Albergar los sensores de distancia**
Soporte para cámara	Vástago y base mecanizados	**Albergar la cámara de video**

Fig. 1: Tabla Resumen de Componentes Mecánicos y Eléctricos

A continuación se muestra una figura donde se pueden identificar los elementos anteriormente mencionados:

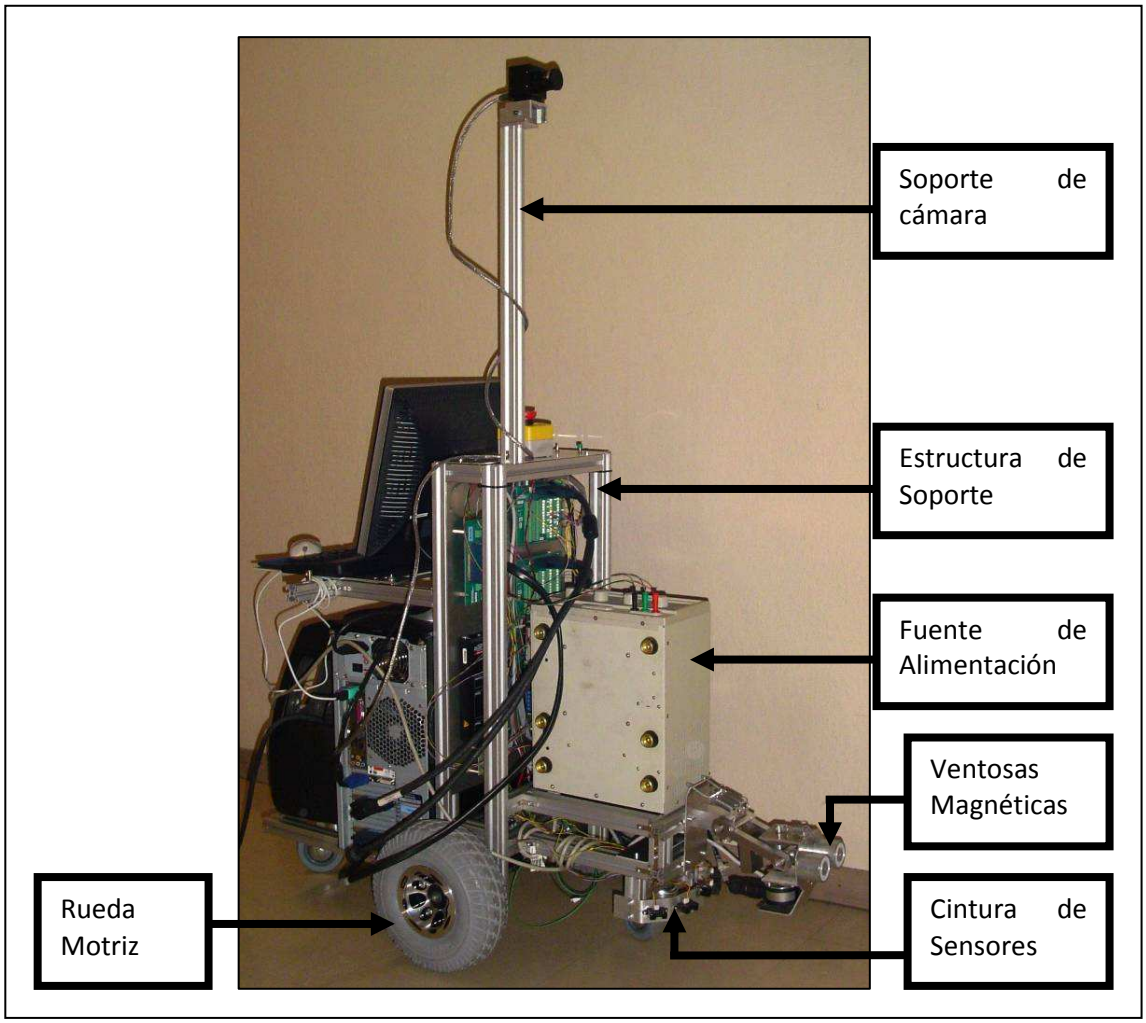

Fig. 2: Vista General de los Componentes Mecánicos y Eléctricos

En esta imagen ya se pueden apreciar los elementos electrónicos que se comentarán a continuación, entre los que resalta el ordenador embarcado. Pero antes de llegar a ese punto, nótense algunos detalles como las ruedas no motrices, que ayudan a soportar la estructura o la alimentación general a través de una línea de 220 v. habitual.

Parte electrónica.

Igualmente que en el apartado anterior, del trabajo de los años precedentes, al comienzo de este proyecto el robot disponía de los elementos resumidos en la siguiente tabla:

COMPONENTE	DESCRIPCIÓN	FUNCIÓN
Ordenador	PC de propósito general con todos sus periféricos	Alberga las tarjetas de adquisición y ejecuta los programas de gobierno
Tarjetas de adquisición de datos	Componentes informáticos que digitalizan información	Obtención y generación de todas las señales correspondientes al robot
Variadores de frecuencia	Componentes que actúan como drivers de los motores	Regulan la velocidad de los motores de las ruedas motrices
Cámara de video	Sensor de imágenes	Captura de imágenes para conocer el entorno del robot
Sensores de distancia	Elemento capaz de obtener la distancia a un objeto	Obtener la distancia a los diferentes obstáculos del entorno del robot
Sensor angular	Elemento que obtiene el ángulo que describe su eje	Medición del ángulo que conforma el robot con respecto a la silla de ruedas
Mando a distancia	Emisora RF de 6 canales	Manejo del robot a distancia
Borneros	Conjunto de elementos de conexionado eléctrico	Alimentar y conectar componentes y tarjetas de adquisición

Fig. 3: Tabla Resumen de Componentes Electrónicos

La mayor parte de estos componentes pueden ser observados en la figura 2. Cabe comentar también que, la cintura de sensores, está compuesta por dos tipos de sensores según su principio de medida: 4 sensores infrarrojos y un ultrasonidos (a partir de ahora IR y US).

En la siguiente tabla se pueden observar las tarjetas de adquisición de datos, del fabricante *National Instruments* (a partir de ahora NI), montadas en el ordenador:

TARJETA	6221	6602.1	6602.2
Modelo	PCI 6221	PCI 6602	PCI 6602
Entradas Analógicas	16	—	—
Salidas Analógicas	2	—	—
E/S Digitales	24	40	40
Timers	—	8	8
Conexión Serie/Par.	—	✓	✓
Puertos USB	—	✓	✓
Conexiones del robot	Motores	Mando a Dist.	Sensor US
	Encoders Ruedas	Sensor Angular	
	Sensores IR	Ventosas	
E/S Libres	8A / 24D	23D	39D

Fig. 4: Tabla Resumen de Características de las Tarjetas de Adquisición

Para ampliar cualquier información referente al desarrollo del robot HAMMI anterior a este proyecto, se pueden consultar las referencias bibliográficas [2], [3] y [4].

Terminando este breve repaso sobre los trabajos acometidos los años anteriores, es necesario decir que existe una serie de programas, estudios y modelos para el gobierno del robot. No presentados aquí, estos trabajos están referenciados en bibliografía y son utilizados para continuar el desarrollo en el presente proyecto.

3. PROGRAMACION DEL ROBOT. Visual C++ y National Instruments

El gobierno del robot se llevará a cabo a través de aplicaciones desarrolladas en lenguaje *Visual C++* y con el entorno de desarrollo el *Microsoft Visual Studio 2005* y bibliotecas de NI.

Se presentará ahora la arquitectura de todas las aplicaciones y los nuevos programas desarrollados durante el presente proyecto.

a) Arquitectura Document-View, Visual Studio 2005 y NI.

Entre las diversas arquitecturas que ofrece el lenguaje y entorno de programación utilizados, se ha elegido la Document-View. Las justificaciones de esta elección son: la herencia de los anteriores proyectos, las recomendaciones de NI para el desarrollo de aplicaciones y la mejor estructuración del programa según dos clases principales.

Estas clases son el documento y la vista, de ahí el nombre de esta arquitectura. En general, en el documento se gestionarán todos los datos del programa mientras que la vista se encargará de todo lo referente al interfaz de usuario. Ambas clases se comunican e incluyen diversos ficheros según lo muestra la siguiente figura:

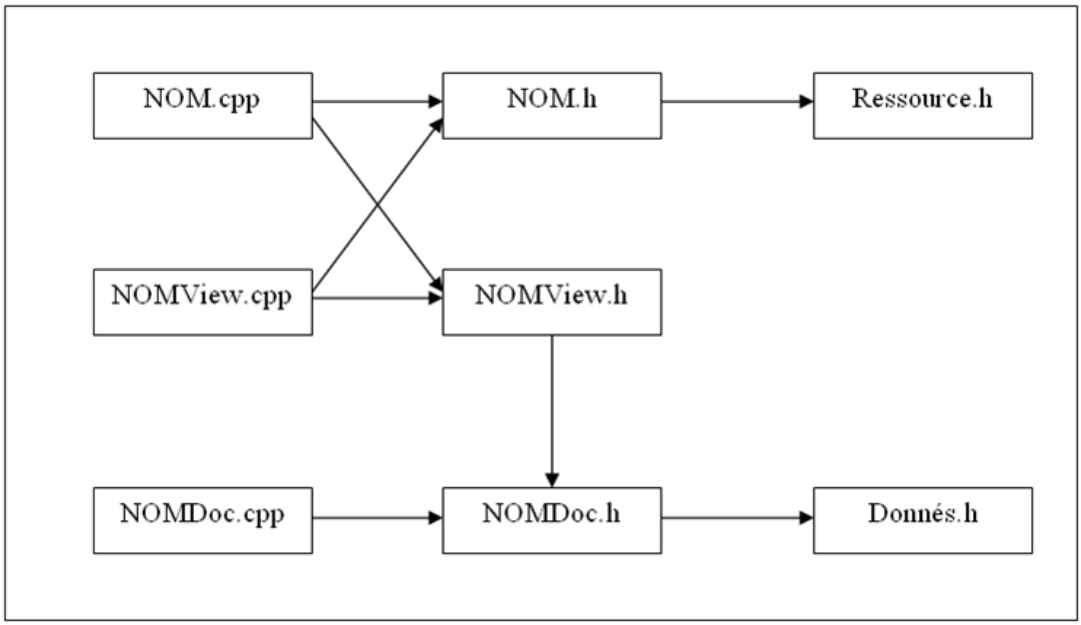

Fig. 5: Arquitectura Document-View en Visual Studio 2005

Esta estructura de archivos es creada automáticamente al generar un proyecto adecuadamente en el entorno de desarrollo *Visual Studio 2005*. Se observan así los ficheros de código *.cpp* que contienen todas las funciones y manejadores y los ficheros *.h* de declaración de funciones y variables. El distintivo *NOM* hace referencia a un nombre genérico elegido por el programador al que se le añadirá el final indicado.

Una vez generado un proyecto Document-View, aparece el entorno de desarrollo de la aplicación como se muestra en la siguiente figura:

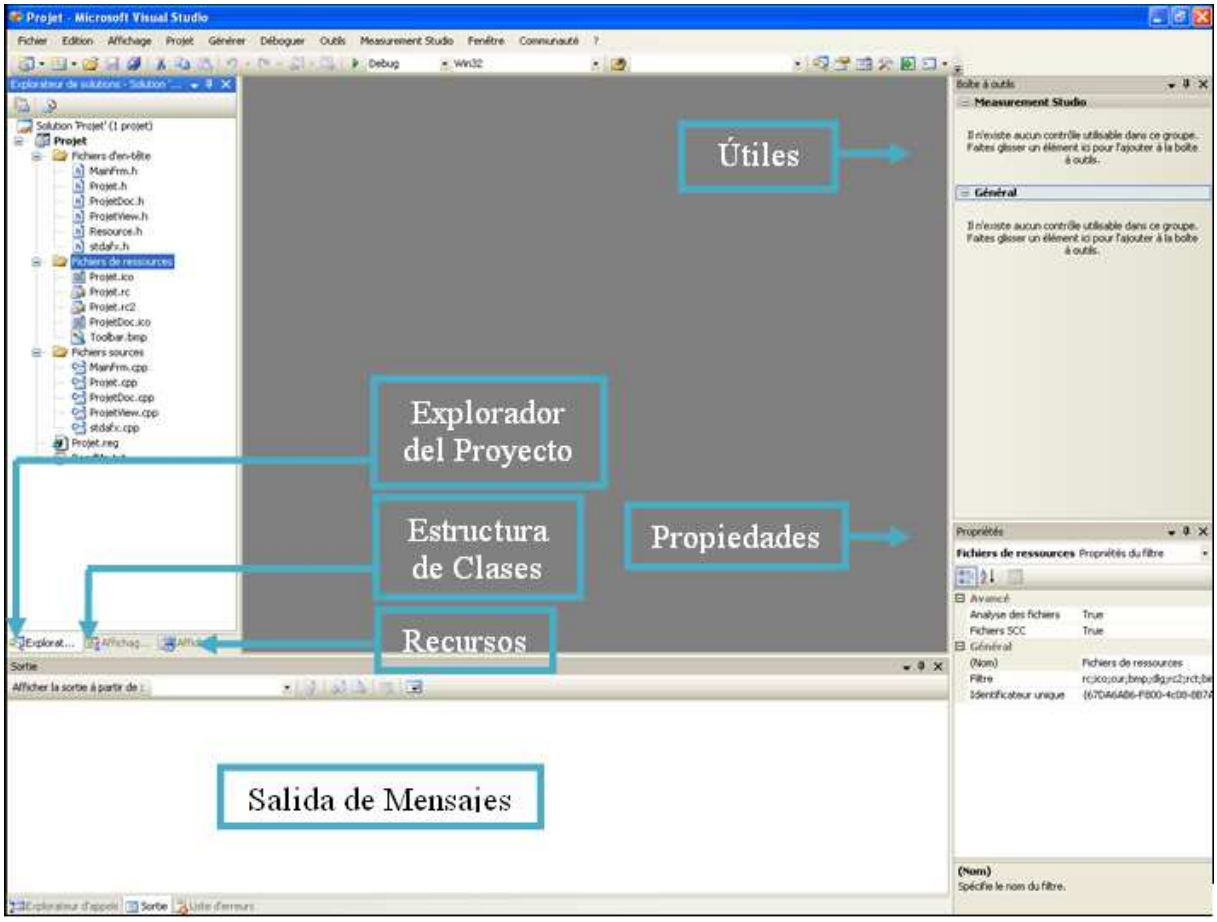

Fig. 6: Entorno de Desarrollo de Visual Studio 2005

Añadido al entorno de desarrollo, NI incluye una serie de bibliotecas que permiten utilizar las tarjetas de adquisición de datos y una serie de recursos exclusivos, tales como clases, elementos gráficos, etc.

En la siguiente imagen se presenta el interfaz de usuario de una aplicación de ejemplo, explicado en el anexo a), desarrollado con elementos de NI:

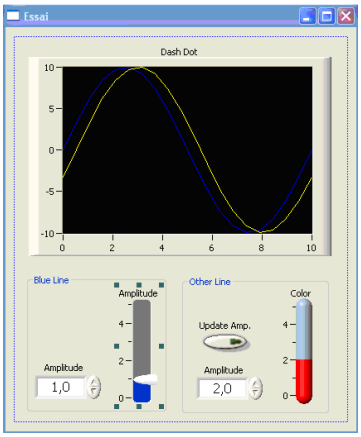

Fig. 7: Aplicación de Ejemplo

Para cualquier duda sobre la arquitectura Document-View y el entorno de desarrollo *Visual Studio 2005*, consúltese la referencia bibliográfica [6]. Si se pretende profundizar en los recursos de NI y su herramienta principal llamada *Measurement Studio* diríjase a la referencia [8].

4. PROGRAMACION DE LA TRAYECTORIA PARA EVITAR OBSTACULOS

b) Programa existente. Robot_HAMMI v2.2

Lo primero de todo es necesario conocer bien lo que se ha desarrollado sobre el robot durante todos los años precedentes.

El programa que dirige el robot ha sido mejorado sucesivamente durante todos los años anteriores creando diferentes versiones. Es por eso que se ha hecho el análisis de la versión más moderna del programa que gobierna el robot. La versión *Robot_HAMMI v2.2*.

Robot_HAMMI v2.2

Este programa tiene como objetivo el gobierno del robot HAMMI para trazar una trayectoria recta entre dos puntos y después describir un arco de circunferencia. Incluye la ley de gobierno, la comunicación con los motores, el tratamiento y estimación de los datos adquiridos por los sensores de distancia así como los ficheros de base (documento, vista, declaraciones, etc.).

Es un programa de tamaño considerable que se compone de 17 clases, 25 ficheros de cabeza y 21 ficheros fuente. Se puede dar cuenta del tamaño del programa con las figuras siguientes:

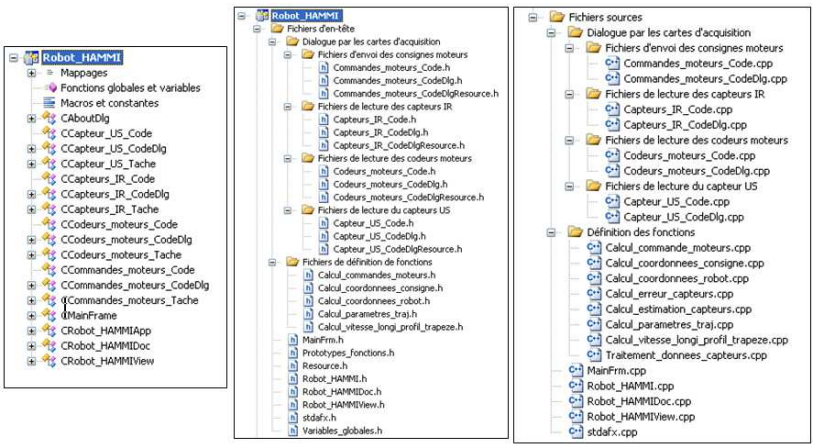

Fig. 8: Clases, ficheros de cabeza y ficheros fuente

Ahora se va a explicar un poco los componentes más importantes de este programa y sus funciones:

La arquitectura que sigue este programa es la documento/vista como ya vimos con anterioridad se concentra en sus clases CNOMDoc y CNOMView.

CRobot_HAMMIView

Esta clase es esencial en nuestro caso ya que ella define la vista de la aplicación. Contiene los bucles de adquisición y tratamiento de datos. La declaración de sus miembros aparece en el fichero ***Robot_HAMMIView.h*** y la definición de sus miembros en el fichero ***Robot_HAMMIView.cpp***.

CRobot_HAMMIDoc

Esta clase gestiona el documento de la arquitectura Doc/View. Contiene por ejemplo las inicializaciones de las variables necesarias. La declaración de sus miembros aparece en el fichero ***Robot_HAMMIDoc.h*** y la definición de los mismos en el fichero ***Robot_HAMMIDoc.cpp***.

Otros componentes muy importantes son:

Prototype_fonctions.h

Fichero de cabeza en el cual son declarados los prototipos de funciones que sirven para calcular las ordenes de velocidad a aplicar en los motores para seguir la trayectoria deseada.

Variables_globales.h

Fichero de cabeza que define todas las variables globales utilizadas en el programa.

CCodeurs_moteurs_Code/CodeDlg

Estas clases permiten efectuar simplemente la adquisición de las señales de los codificadores.

CCommandes_moteurs_Code/CodeDlg

Estas clases permiten efectuar el envío de las órdenes a los motores.

Traitement_donnees_capteurs.cpp

Este fichero realiza el filtro numérico y el calibrado de las medidas.

A continuación la interfaz de usuario de este programa existente al comienzo del proyecto con sus partes explicadas:

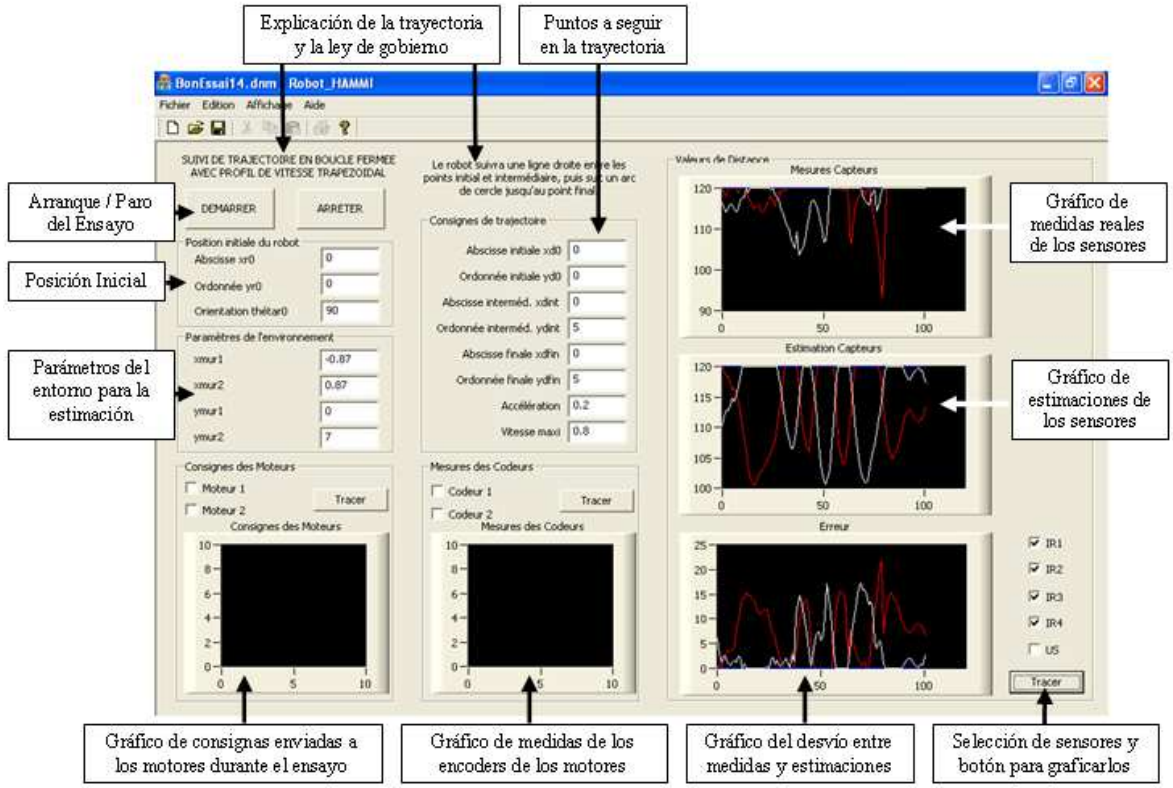

Fig. 9: Interfaz de usuario de Robot_HAMMI v2.2

Una vez que se ha presentado el programa existente se puede describir las mejoras que se han hecho en el programa.

c) Desarrollo del nuevo programa. Robot_HAMMI v3.2

En este apartado se resume todo lo que concierne a la programación del programa que se encuentra en el robot. Los cambios realizados sobre el programa anterior que ha sido descrito en la sección anterior nos han permitido llegar a la nueva versión, el *Robot_HAMMI v3.2.*

Dentro de la versión precedente cuando el robot encuentra un obstáculo este se paraba. Esta nueva versión se centra en la trayectografía para esquivar los posibles obstáculos que el robot se pueda encontrar y así poder continuar con su camino. También se ha adjuntado la nueva funcionalidad de poder guardar los datos registrados en la matriz « m_donnees » en un fichero externo a la aplicación para su posterior tratamiento.

Es necesario decir que todos los trabajos realizados sobre el robot han sido hechos sin tener en cuenta la silla de ruedas, es decir, los movimientos son programados para cuando el robot se encuentra solo.

Existen algunas funcionalidades de la interfaz usuario de la versión anterior (v2.2) que no han sido utilizadas en este proyecto pero aun así no han sido suprimidas de la nueva interfaz ya que puede que en

posteriores proyectos sean de nuevo útiles. Nos hemos limitado a modificar un poco esta interfaz con los útiles necesarios para la nueva funcionalidad de guardado de datos.

A continuación la interfaz de usuario principal de la versión actual, v3.2:

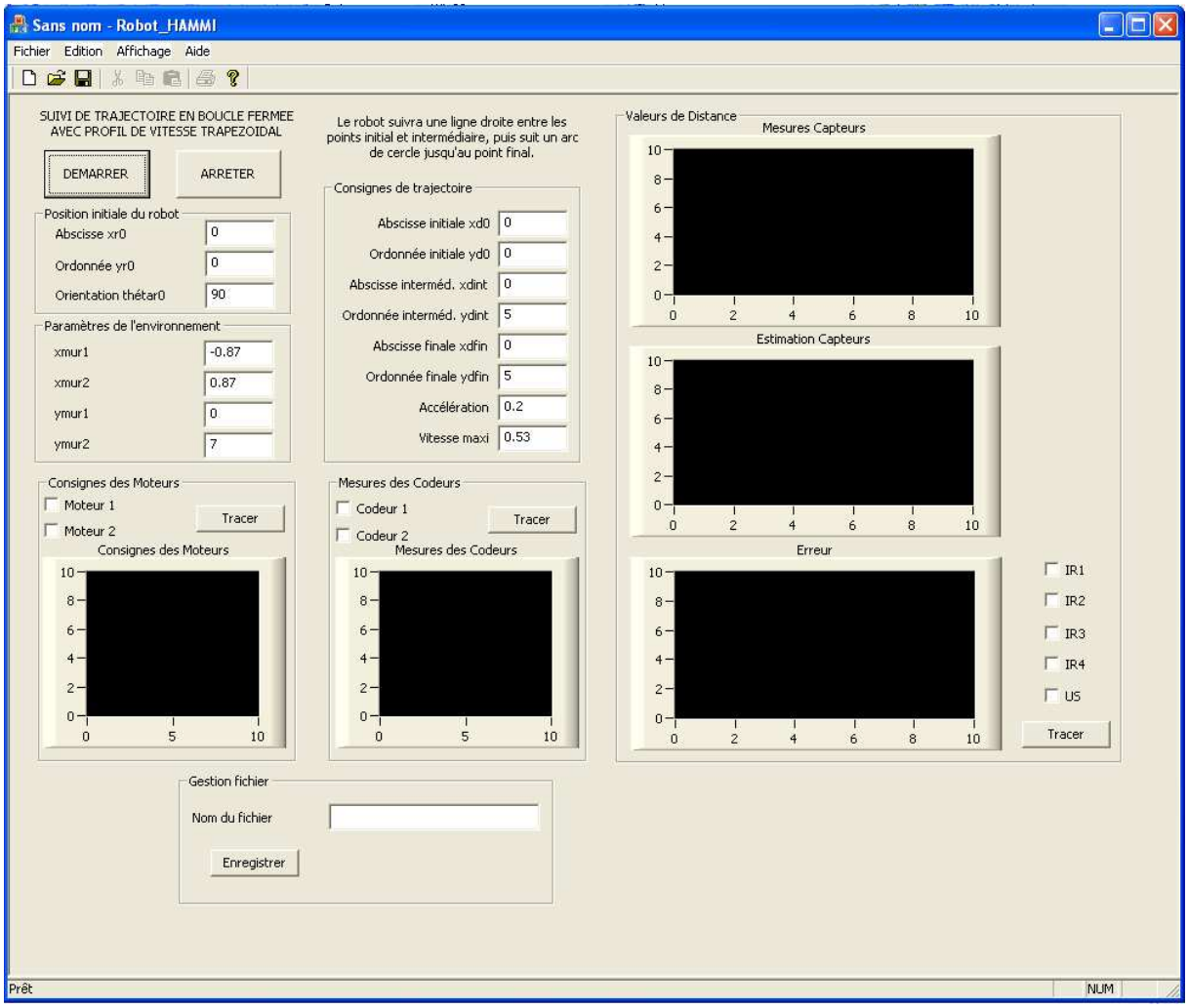

Fig. 10: Interfaz de usuario del Robot_HAMMI v2.3

Metodología seguida para evitar los obstáculos

Lo primero que se ha de hacer para evitar los obstáculos es ver donde se pueden encontrar dichos obstáculos. Después se debe calcular si se puede chocar con esos obstáculos si se continúa sin cambiar la trayectoria que el robot sigue en ese momento.

Para el objetivo de ver si el robot puede chocar con los obstáculos se va a calcular la componente horizontal (en el eje Xr) de la medida que se obtiene mediante los sensores de distancia. Si la componente horizontal es menor que la distancia entre el sensor y la parte más externa del robot, el robot chocará contra el obstáculo.

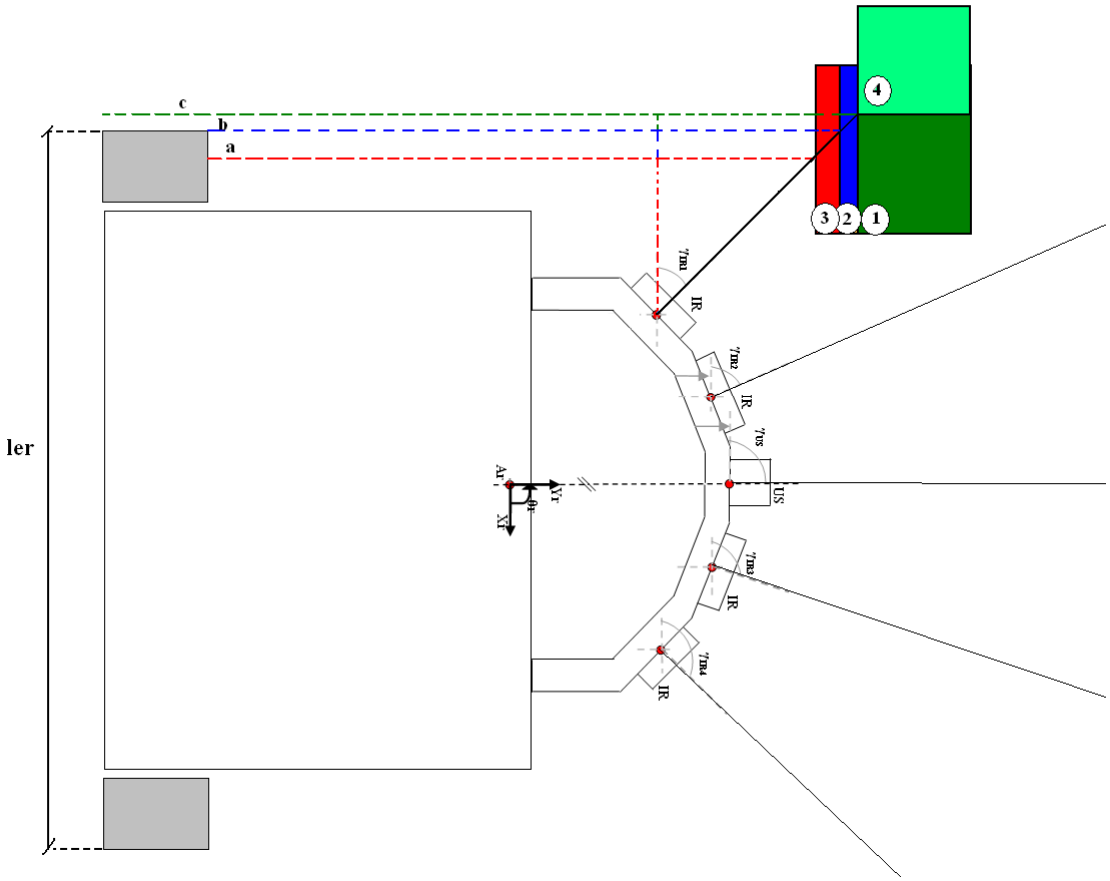

Fig. 11: Robot con unos obstáculos

En la figura 11 se representan dos situaciones diferentes. Primero, un mismo objeto que cambia su posición (1,2 y 3) cuando el robot avanza en línea recta. Y segunda, un objeto en otra posición diferente a la primera situación (numero 4).

En la primera situación se ven tres posiciones diferentes (1,2 y 3) y las medidas que el sensor IR1 (el primero de la izquierda) hace de un mismo objeto cuando el robot avanza en línea recta.

- Se puede ver que en la primera posición del objeto (1) la medida que el robot toma del objeto dice que el robot puede pasar sin chocar con el objeto (línea verde, c)
- La medida en la segunda posición (2) detecta que se está en el límite para pasar (línea azul,b).
- Cuando se está suficientemente cerca del objeto (3), la componente horizontal de la medida muestra que el robot chocará con el objeto si no tuerce (línea roja, a).

En la segunda situación se tiene el objeto en la posición 4. Se puede ver que el sensor da la misma medida para el objeto en la posición 4 que el objeto de la primera situación en la posición 1. La diferencia es que como ya hemos visto, cuando el robot se acerca al objeto de la primera situación la medida del sensor cambia hasta detectar que el robot debe torcer. Cuando el robot se acerca al objeto de la segunda situación (posición 4) la medida no va cambiar y por tanto el robot puede pasar sin cambiar su camino.

A partir de esta distancia limite horizontal que el robot no debe sobrepasar se va a calcular la medida limite de cada sensor de distancia. Si el robot respeta esas medidas limite no tendrá ningún problema con los objetos que se encuentran en su entorno. Para alcanzar este objetivo se necesitan los parámetros geométricos de los sensores y sus valores. Los podemos ver en las figuras 12 y 13.

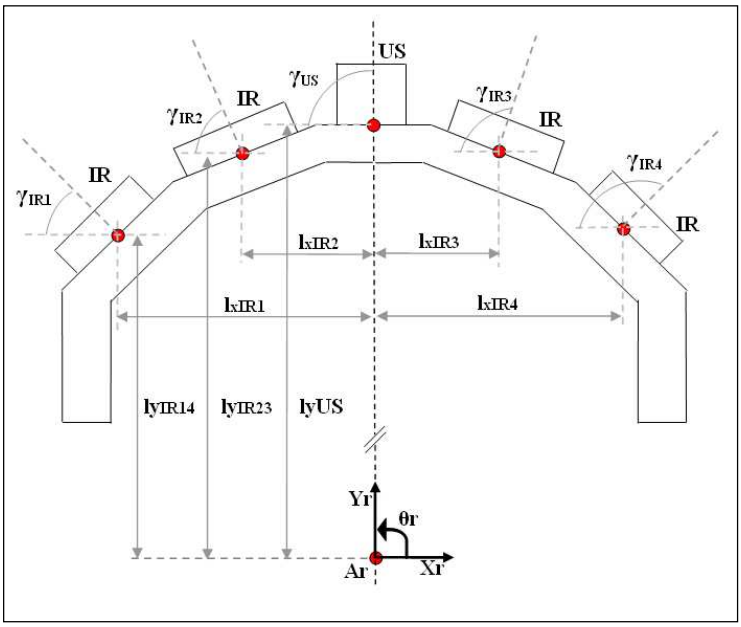

Fig. 12: Parámetros geométricos de los sensores de distancia

Los valores numéricos de todos esos parámetros son listados en la tabla de la figura 34:

Parámetro	Valor
γIR1	50°
γIR2	72°
γIR3	110°
γIR4	135°
γUS	90°
lyIR14	0,329 m
lyIR23	0,355 m
lyUS	0,363 m
lxIR1	-0,091 m
lxIR2	-0,044 m
lxIR3	0,042 m
lxIR4	0,090 m

Fig. 13: Tabla de Valores de los parámetros geométricos

Una vez que se tienen los parámetros geométricos queda calcular el valor de la medida limite **IRxL** por medio de unos simples cálculos que se van a ilustrar con la figura 14 las ecuaciones siguientes:

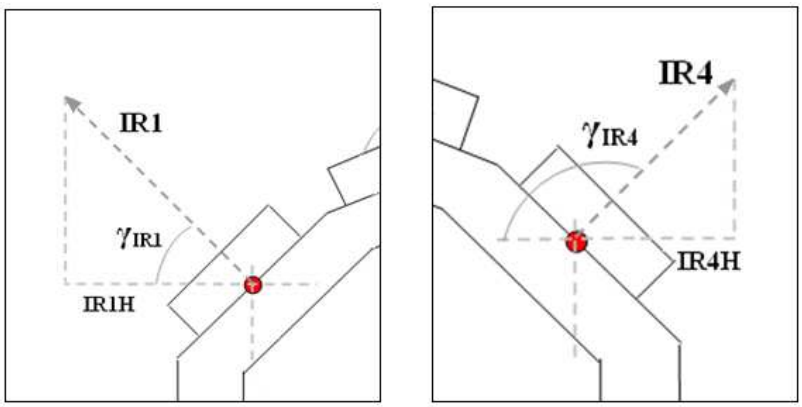

Fig. 14: Componentes de la medida. Caso IR1 et IR4.

Se pueden ver 2 casos diferentes:

Un caso para los sensores IR1 et IR2:

$$IR1 = \frac{IR1H}{\cos \gamma_{IR1}} \quad \text{(ec. 1)}$$

y el otro para los sensores IR3 et IR4:

$$IR4 = \frac{IR4H}{\cos(180 - \gamma_{IR1})} \quad \text{(ec.2)}$$

Para obtener el valor límite se debe calcular a la distancia mínima horizontal que debe haber entre el sensor y la parte más externa del robot.

$$IR1HL = \frac{ler}{2} - IxIR1 \quad \text{(ec.3)}$$

Donde ler es la longitud del eje del robot y tiene un valor de 0.5 m.

Después, cuando se ha utilizado el robot se ha visto que se debe incluir un valor adicional a esa distancia para el buen funcionamiento del robot cuando esquiva los obstáculos. Ese valor es de 0.1m para los captores IR1 e IR2 y 0.2m para IR3 e IR4.

Por tanto, se tienen los siguientes resultado para las medidas limites de cada sensor:

$$IR1L = \frac{IR1HL}{\cos \gamma_{IR1}} = \frac{\left(\frac{ler}{2} + 0.1 - IxIR1\right)}{\cos \gamma_{IR1}} \quad \text{(ec.4)}$$

$$IR2L = \frac{IR2HL}{\cos \gamma_{IR2}} = \frac{\left(\frac{ler}{2} + 0.1 - IxIR2\right)}{\cos \gamma_{IR2}} \quad \text{(ec.5)}$$

$$IR3L = \frac{IR3HL}{\cos(180 - \gamma_{IR3})} = \frac{\left(\frac{ler}{2} + 0.2 - IxIR3\right)}{\cos(180 - \gamma_{IR3})} \quad \text{(ec.6)}$$

$$IR4L = \frac{IR4HL}{\cos(180 - \gamma_{IR4})} = \frac{\left(\frac{ler}{2} + 0.2 - IxIR4\right)}{\cos(180 - \gamma_{IR4})} \quad \text{(ec.7)}$$

Ya que el sensor ultrasónico (US), como se ha visto en la figura 32, no tiene componente horizontal el valor límite es el que le queramos dar. Se ha elegido 60cm. USL=60cm.

Modificación del código para evitar obstáculos

El código creado para evitar obstáculos se basa en programar una serie de condiciones teniendo en cuenta la relación entre las medidas obtenidas por los sensores de distancia y los limites calculados en el apartado anterior. En principio esas relaciones que se han programado son muy generales para después modificarlas si es necesario con las excepciones que se puedan encontrar.

Al principio las condiciones que se han programado son:

1. Si la medida del sensor US es menor que el limite USL, el robot debe verificar donde hay más espacio por el cual evitar el obstáculo. Para ese fin el programa calcula la distancia horizontal que los sensores izquierdos y derechos dan y las compara. Todo eso ha sido programado en la primera versión del programa (*Robot_HAMMI v3.1*) pero experimentalmente se ha descartado la utilización de esta condición ya que la medida obtenida por el sensor US es bastante variable y por tanto poco fiable. También, se ha visto que no se necesita esa medida porque con las medidas de los sensores infrarrojos es suficiente para detectar los obstáculos. Si en proyectos futuros se tiene un funcionamiento mejor del sensor se puede tratar de utilizar esta programación que es presentada en el anexo b).

2. Si la medida del sensor IR es menor que el limite IRL quiere decir que hay un obstáculo y por tanto el robot debe torcer al lado opuesto. Para conseguir esto se ha programado que la rueda opuesta al sensor que ha detectado el objeto disminuya su velocidad hasta que la medida del sensor sea mayor que su límite. En consecuencia el robot torcerá al lado opuesto hasta que el robot evite el

obstáculo. Esta condición y respuesta es igual para los cuatro sensores IR. Cada vez que se cumple esta condición se muestra un aviso por pantalla que dice en qué lado se encuentra el objeto.

3. Hay otra posibilidad que nuestro programa contempla. Si al mismo tiempo hay dos sensores opuestos (derecha e izquierda) que su medida es menor que su límite, el robot se parará porque no habrá espacio para pasar y mostrará un aviso por pantalla.

Se puede encontrar en el anexo b) todo el código que se ha escrito para conseguir lo que se ha descrito en esta sección con comentarios para una mejor comprensión.

Modificación del código para guardar los datos

Para una posterior validación del modelo creado hace falta adjuntar en el programa un código que sirve para guardar los datos adquiridos durante la realización de los tests.

Para más comodidad, el software que se va a utilizar a para el análisis de los datos es Matlab. Por tanto, esos datos deben estar guardados en un fichero externo que pueda ser leído por el software. Una vez que se tiene el documento con los datos se puede abrir y analizar todas las veces que se quiera y de la manera que se quiera.

Ese fichero va a contener todas las informaciones que se han guardado en la matriz “m_donnees” de nuestro programa. Ese fichero es un fichero columna (que se puede abrir con el Notepad por ejemplo) en el cual todos los datos son guardados unos detrás de otros.

En la figura siguiente se puede ver los recursos que se han adjuntado a la interfaz usuario del programa *Robot_HAMMI v2.2* para poder conseguir el objetivo y la nueva interfaz usuario:

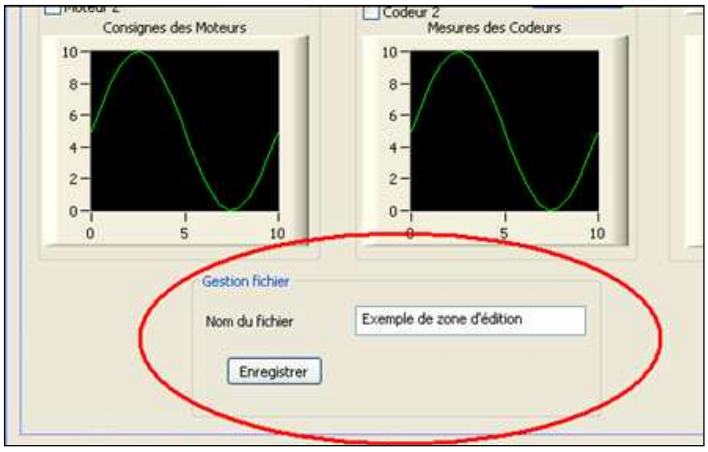

Fig.15: Parte interfaz usuario Robot_HAMMI v3.2.

Se puede ver que en la parte *Gestion fichier* hay un espacio para escribir el nombre del fichero que se quiere dar al nuevo fichero. Hace falta también pulsar sobre el botón *Enregistrer* para hacer efectivo dicho registro.

Importante: El nombre del fichero no debe contener puntos porque provocan un error cuando el programa de matlab intenta trabajar con él.

Si se mete en como nombre del fichero un nombre que ya existe aparece una nueva ventana en la que se previene de la existencia de un fichero con el mismo nombre y se pregunta si se quiere remplazar el fichero existente. También se muestra esta ventana si el fichero está abierto por algún otro programa.

Para gestionar todo esto se ha creado una nueva clase *CErreur_Nom_Fichier* con todos sus componentes, recursos y código necesarios. Se puede ver en la figura siguiente la venta que aparece para gestionar esa tarea.

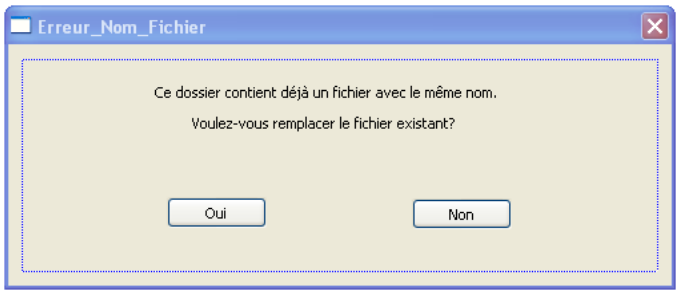

Fig. 16: Ventana para gestionar el guardado de ficheros con el mismo nombre.

Si vamos al anexo c) podemos encontrar todo el código creado para conseguir el guardado de la matriz m_donnees en el fichero así como el código para gestionar los errores mencionados arriba.

d) Creación de un programa para el análisis de los datos en MATLAB

Una vez que se han guardado los datos en un fichero hace falta crear un programa para el análisis de los datos que contiene.

A partir del programa *exploi_mesu_ens* creado por S.Rubrecht en Junio de 2006 se ha escrito un programa en Matlab adaptado a nuestras necesidades permitiéndonos explotar los datos. Ese programa se llama ***exploi_mesu_ens_ROBOTHAMMI_v32.m***. Por supuesto depende del número de variables guardadas para poder reconstruir los vectores de valores consecutivos de Omega_1, Consigne_Mot_1, etc. Este programa ha sido creado ya que es mucho más fácil explotar los resultados en Matlab que en Visual Studio.

Una vez que se han reconstruido los vectores se ha escrito el código para tener los gráficos que se necesitan para el posterior análisis de resultados. Ese código puede ser modificado de forma fácil para adaptarlo a futuras necesidades.

Cuando se ejecuta el programa el pide el numero de variables que hay en el fichero a explotar. Si dejamos vacío este espacio el programa coge 38 por defecto. Después el pide el nombre del fichero a explotar y muestra todos los gráficos programados.

En el anexo d) se puede ver el código completo con comentarios del programa ***exploi_mesu_ens_ROBOTHAMMI_v32.m***.

e) Validación del método empleado

Después de haber hecho la programación queda hacer la validación de aquello que se ha programado. Para conseguirlo se han realizado varios test, se han guardado los datos necesarios que sirven a analizar los resultados y por tanto a validar el método de esquivar obstáculos.

Test realizados

Para ver si el programa realizado ejecuta el objetivo para el cual ha sido concebido se han creado 5 tests diferentes:

Test 0: Este primer test tiene como fin solamente ver el funcionamiento de las medidas tomadas por los sensores sin ningún objeto que perturbe las medidas.

El robot es situado en el medio del pasillo sin ningún objeto en su camino. El robot debe ir recto.

Fig.17: Test0 Robot_HAMMI v3.2

Test1: Primer tipo de los test realizados para ver si el robot evita bien los obstáculos y estudiar los datos recibidos.

Se debe decir que aquí cuando se habla de derecha e izquierda son las orientaciones relativas al robot en el sentido de su avance.

El robot es situado en el lado izquierdo del pasillo. En el pasillo hay 2 obstáculos. El primero obstáculo esta a la izquierda, en frente del robot pero solo la parte izquierda del robot se encontraría con el obstáculo si no torciera. El segundo obstáculo esta a la derecha del pasillo.

Fig. 18: Test1 Robot_HAMMI v3.2

Test2: El robot está situado al lado derecho del pasillo. En el pasillo hay dos obstáculos. El primero está a la derecha, en frente del robot pero solo la parte derecha del robot se encontraría con el obstáculo si este no torciera. El segundo obstáculo esta a la izquierda.

Fig. 19: Test2 Robot_HAMMI v3.2

Test3: El robot está situado al lado derecho del pasillo. En el pasillo hay un obstáculo totalmente en frente del robot. El robot lo debe evitar.

Fig. 20: Test3 Robot_HAMMI v3.2

Test4: El robot está situado al lado izquierdo del pasillo. En el pasillo hay un obstáculo totalmente en frente del robot. El robot lo debe evitar.

Fig. 21: Test4 Robot_HAMMI v3.2

Análisis de los resultados

Antes de comenzar con el análisis de los resultados se debe tener en cuenta alguna consideración.

Si vamos al proyecto que se encuentra con la referencia [5] se encuentra que el autor dice que el límite de percepción de los sensores IR es de entre 20 y 120 cm y para el sensor US entre 20 y 600cm.

El dice también que básicamente el error máximo cometido en los datos no sobrepasa los 15cm y normalmente por debajo. Esta variabilidad puede ser causada por la variabilidad en el ángulo que forman los sensores con el suelo causadas por el movimiento del robot. Se debe ver si esta variación de 15cm de las medidas puede influir en los test realizados.

Otra cosa importante es que todos los datos recogidos son de test que han sido satisfactorios.

Durante casi todos los test se ha grabado en video el comportamiento del robot para poder volver a ver el comportamiento del robot a la misma vez que los resultados y por lo tanto hacer un mejor análisis de estos.

A continuación se va a analizar los tests que muestran mejor el comportamiento del robot.

TEST0.1_v3.2

Se han realizado 4 test de tipo 0. Aquí se muestran los resultados del TEST0.1_v3.2 como ejemplo para analizar el comportamiento de los sensores y la estabilidad de sus medidas:

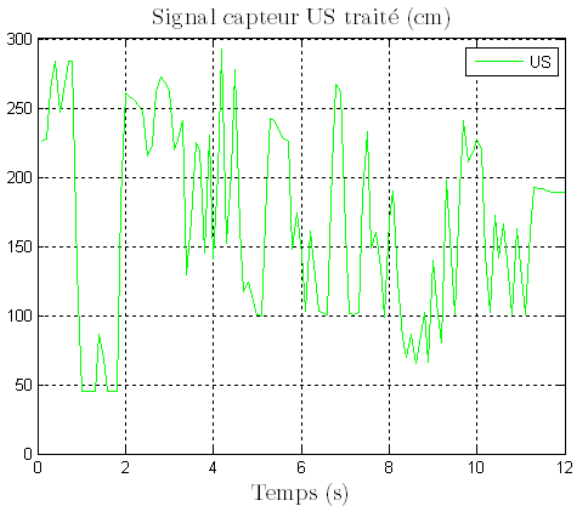

Fig. 22: Señal sensor US tratada Test0.1_v3.2

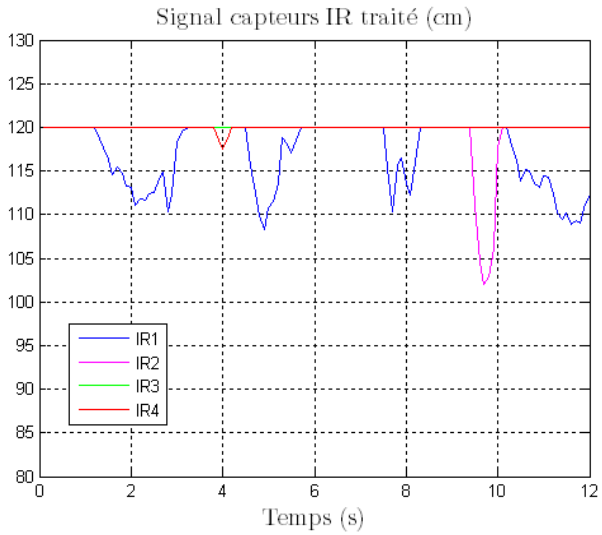

Fig. 23: Señal sensor IR tratada Test0.1_v3.2

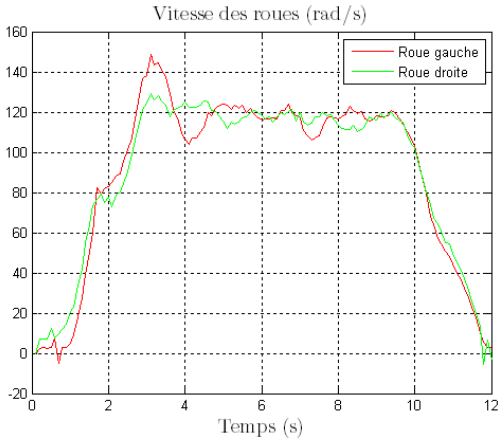

Fig.24: Velocidad de las ruedas Test0.1_v3.2

Aquí se puede ver sobre la figura 22 que la medida obtenida por el sensor US es bastante variable (de 45cm a 390cm) y por tanto poco fiable. En un pasillo sin obstáculos esa medida debería ser linealmente decreciente cuando el robot se aproxima a la pared final del pasillo. Se ha observado el mismo comportamiento en todos los otros test realizados y de todos los tipos.

Por esta razón no se ha tenido en cuenta este sensor (US) para la programación de la trayectoria.

En la figura 23 se ve que IR3 e IR4 no cambian y que IR1 e IR2 cambian un poco. La situación ideal sería que IR1 e IR2 no cambiaran tampoco porque no hay ningún obstáculo pero se ha comentado ya que podía haber una variación de 15cm que es más o menos la variación observada.

Para ir recto las velocidades de las ruedas del robot deben ser iguales. En la figura 46 se puede observar que las velocidades son prácticamente iguales y por tanto el robot irá derecho.

Se debe notar también las unidades de la velocidad de las ruedas, en rad/s, que se han tomado del año anterior. Esto debe ser un error porque la velocidad de rotación de las ruedas llega a los 140rad/s que corresponden a 21m/s lo cual es mucho. Puede que la medida sea en rev/min la cual correspondería a 2.2m/s que es una medida más razonable para nuestro robot.

Test1.6_v3.2

Se han realizado 10 test de tipo 1(7 con video), 6 de tipo 2(5 con video), 5 de tipo 3 (todos con video) y 6 de tipo 4 (5 con video). Se muestran aquí los resultados del test1.6_v3.2 como ejemplo que sirve para mostrar el comportamiento del robot de forma general. Se ha escogido este test porque es uno de los más claros:

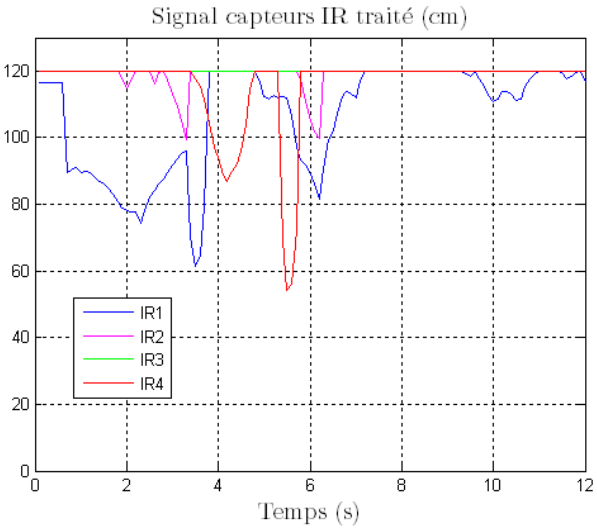

Fig. 25: Señal sensores IR tratada Test1.6_v3.2

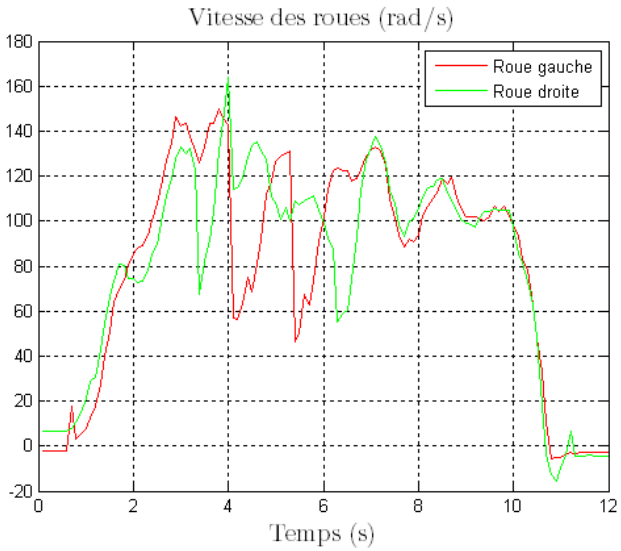

Fig. 26: Velocidad de las ruedas Test1.6_v3.2

Podemos observar en las figuras 25 et 26:

- Instante 3,5: El sensor IR1 detecta el objeto izquierdo y la velocidad de la rueda derecha disminuye para torcer por el lado derecho y evitar el obstáculo.
- Instante 4,25: El sensor IR4 detecta la pared derecha y la velocidad de la rueda izquierda disminuye para torcer por el lado izquierdo y evitar chocar contra el muro.
- Instante 5,5: El sensor IR4 detecta el objeto derecho y la velocidad de la rueda izquierda disminuye para torcer por el lado izquierdo y evitar el obstáculo.
- Instante 6,25: El sensor IR1 detecta la pared izquierda y la velocidad de la rueda derecha disminuye para torcer por el lado derecho y evitar chocar contra el muro.
- Después del instante 7: Los sensores no detectan nada y las velocidades de las ruedas se igualan para ir recto.

Podemos decir que la reacción del robot es buena cuando encuentra un obstáculo. Por tanto, estamos satisfechos del comportamiento del robot.

Para más resultados de test se puede ir a los ficheros de la carpeta *TEST_ROBOT_HAMMI_v3.2* y ejecutar dichos ficheros con el programa de matlab desarrollado *exploi_mesu_ens_ROBOTHAMMI_v32.m*

5. CONCLUSIONES Y TRABAJOS FUTUROS

Para terminar se presentan los objetivos alcanzados durante la primera y segunda parte de este proyecto y se proponen los trabajos futuros que pueden ser buenos desarrollar en futuros proyectos.

En la primera parte del proyecto se ha analizado los proyectos y trabajos de años anteriores con el objetivo de adquirir un conocimiento del desarrollo ya existente y del funcionamiento del robot HAMMI.

También, se ha explicado el aprendizaje y el progreso que se ha hecho en cuanto a la programación (arquitectura, recursos NI,…), el lenguaje (Visual C++) y el software (Microsoft Visual Studio 2005 y Measurement Studio) que hace falta utilizar. Se han unido las informaciones de varias fuentes y se ha sintetizado aquí todo aquello que se ha creído más interesante para este proyecto. Se han explicado los procedimientos para construir las aplicaciones y los recursos que se disponen para hacerlas. Finalmente se ha mostrado una de las aplicaciones realizadas para reforzar el aprendizaje. Todo ese trabajo ha sido muy importante para crear una base de conocimiento la cual ha permitido trabajar en el robot y alcanzar el objetivo propuesto en el proyecto.

En la segunda parte se ha hecho todo el estudio directamente sobre el robot. Se ha comenzado el desarrollo de la aplicación que permite controlar el robot en los entornos con obstáculos. Este trabajo ha sido terminado de manera satisfactoria con una aplicación en la cual el robot HAMMI es capaz de atravesar un pasillo con obstáculos sin chocar con ninguno.

Otros trabajos que se han hecho de manera satisfactoria es el desarrollo de las aplicaciones en Visual Studio 2005 y Matlab para el análisis de los datos y el comportamiento del robot.

Para los trabajos futuros hace falta continuar con el desarrollo de este primer programa de evitar obstáculos para completarlo. Desarrollos futuros pueden concernir testar el robot en diferentes entornos en los cuales el robot se desplace, crear el código que sirva para encontrar el camino que el robot sigue una vez que el obstáculo ha sido esquivado por medio de la localización, etc.

En el informe de la referencia [5] el autor propone como trabajo futuro la implantación de un filtro Kalman sobre el robot. Es una buena idea ya que las medidas de los sensores serán menos variables y por tanto el robot podrá evitar mejor los obstáculos.

Para terminar se puede afirmar que se han alcanzado todos los objetivos y por tanto el proyecto ha sido finalizado de forma exitosa.

6. BIBLIOGRAFIA

[1] Rapport de l'IReSP (Institut de Recherche en Santé Publique) « LE HANDICAP, NOUVEL ENJEU DE SANTE PUBLIQUE » : http://www.cnrs.fr/infoslabos/appels-offres/docs/Appel_a_projets_handicap_IReSP_2007.pdf

[2] Optimisation mécanique et programmation d'un robot mobile d'assistance aux personnes handicapées : le robot HAMMI ; Florian GENEST ; ENSAM Paris 2008.

[3] Robotique mobile pour l'assistance aux personnes handicapées : conception mécanique et implantation électronique ; rapport final de PJE ; Adrien VENGEANT ; ENSAM Paris 2007.

[4] Partie électronique du robot HAMMI ; rapport final de PJE ; Aurélien SIXTE ; ENSAM Paris 2007.

[5] Fusion de données : Application à un robot mobile. Le robot HAMMI; Javier SAÉZ CARDADOR; ENSAM Paris 2009.

[6] Visual C++ 6 ; Ivor Horton; Wrox Press 1999.

[7] Robot HAMMI : programmation du robot pour l'assistance aux personnes handicapées ; rapport final de PJE ; David NUNEZ ; ENSAM Paris 2007.

[8] Tutorial "Creating a Measurement Studio for Visual C++ 6.0 Project" dans le site de National Instruments : http://zone.ni.com/devzone/cda/tut/p/id/3177.

Rapport en Français

1. TABLE DES MATIÈRES

1. TABLE DES MATIÈRES 31
2. INTRODUCTION 33
3. OBJECTIF DU PROJET 34
4. ÉTAT D´AVANCEMENT DU ROBOT HAMMI 35
a) Partie mécanique et électrique 35
• Structure support 35
• Autres supports 36
• Ventouses électromagnétiques 36
• Roues motrices 37
• Source d´alimentation 37
• Attache fauteuil 37
b) Partie électronique 38
• Calculateur 38
• Cartes électroniques 38
• Bornier analogique 38
• Variateurs 39
• Capteurs 39
• Caméra 39
• Codeur angulaire 40
• Coup de poing (arrêt d´urgence) 40
• Télécommande 40
5. PROGRAMMATION 41
a) Visual C++ 41
• Structure d´un programme Windows 41
• MFC (Microsoft Foundation Classes) 42
• Les Classes 43
• Le concept Document/Vue 43
b) Visual Studio 2005 46
• Créer un nouveau project 46
• Concevoir une interface d´usager 49
• Architecture Document/Vue dans Visual Studio 53

c) Measurement Studio de National Instruments 54
6. PROGRAMMATION DE LA TRAJECTOIRE POUR EVITER OBSTACLES 55
a) Programme existant. Robot_HAMMI v2.2 55
b) Développement du nouveau programme 57
• Méthode d´évitement d´obstacles 57
• Modifications du code pour éviter obstacles 61
• Modifications du code pour enregistrer les données 61
c) Création du programme pour l'analyse des données 63
d) Validation de la méthode 63
7. CONCLUSION ET TRAVAUX FUTURS 69
8. BIBLIOGRAPHIE 70
9. ANNEXES 71
a) Création d'un projet measurement studio 71
b) Code ajouté pour eviter les obstacles 77
c) Code ajouté pour enregister les données 81
d) Programme crée pour le analyses des donnees 86

2. INTRODUCTION

En 2007, la population française avec une déficience motrice quelconque est de plus de 10 %, en atteignant à environ 21% si l'on parle d'autres incapacités. Dans le cas de la communauté européenne on parle aussi de 10% de personnes handicapées sur la totalité de sa population (Voir la référence [9] pour plus d'information sur ces données).

Alors que solidarité et entraide reviennent sur le devant de la scène politique et scientifique et que depuis longtemps la technologie a été développée au service de l'humanité et en satisfaisant une partie de ses besoins, il paraît naturel de faire profiter les personnes handicapées des nombreuses avancées technologiques en robotique et automatique. C´est dans cet esprit que se situe ce projet.

Le LMSP - Laboratoire de Mécanique des Systèmes et des Procédés - de Arts et Métiers ParisTech, avec la collaboration de la société HanditecAM, propose un projet intitulé : Robotique mobile pour personnes handicapées. Ce projet a pour but de concevoir une plateforme de développement qui permettra d'aider les personnes présentant une déficience motrice ou sensorielle en améliorant leur vie quotidienne.

Ce travail à conduit depuis 2006 (1er prix concours Garangeat en 2006) à la conception et la réalisation d'un robot pousseur, dont la principale fonction est la motorisation d'un fauteuil roulant manuel. C´est-à-dire, le robot est capable de s´adapter à n´importe quel fauteuil roulant conventionnel, sans nécessiter de grandes modifications sur ce dernier. Ce robot nommé HAMMI (HanditecAM Arts et Métiers Motorisation Intelligente) est pilotable grâce à une télécommande, mais doit pouvoir se déplacer en autonomie dans un environnement connu. De plus, il est également en projet de lui ajouter un bras manipulateur capable de porter des charges relativement lourdes comme les bagages de la personne assistée.

On commence par la description des objectifs fixés pour ce projet. Ensuite, on décrira les fonctionnalités et des composants actuels sur le robot et on continuera en présentant le travail déroulé sur la plateforme de programmation du robot. Enfin, on exposera les conclusions, les objectifs accomplis et les travaux futurs.

3. OBJECTIF DU PROJET

Avant de présenter l´état d´avancement du robot HAMMI au début du projet on va exposer les objectifs que ce projet doit satisfaire.

Comment on peut voir dans le titre de ce projet, le travail qu'on doit faire est de la programmation de la trajectoire du robot. Ce projet va se concentrer sur la programmation du robot pour éviter les objets qui peuvent se trouver sur la trajectoire que le robot suit. Tout cela sera possible grâce à l'utilisation des travaux développés les années précédentes. Un exemple peut être l´utilisation de l´information fiable obtenue par les capteurs.

Pour atteindre cet objectif on va se familiariser avec toutes les ressources et composants qu´on doit maîtrisé pour manipuler et interagir avec le robot. De telle manière, on assimilera toutes les informations nécessaires et on commencera à essayer des logiciels sur le robot.

Plus précisément, on va apprendre à utiliser le langage de programmation ***Visual C++,*** le logiciel de développement ***Microsoft Visual Studio 2005***, la bibliothèque ***Measurement Studio de National Instrument*** et en général, toute l'architecture et éléments du robot.

4. ÉTAT D´AVANCEMENT DU ROBOT HAMMI

La idée du créer le robot HAMMI viens d´une collaboration entre le LMSP de l´ENSAM Paris et HanditecAM, qui est une fondation impliqué en le développement de divers projets pour promouvoir les technologies au fin de servir a las personnes handicapées.

Les années précédentes, plusieurs projets ont déjà eu lieu sur le même sujet dans des domaines techniques différents comme la programmation, la cinématique, l'électronique et la mécanique. On arrive donc maintenant à un stade relativement avancé de l'étude.

Pour expliquer l'état d´avancement du robot HAMMI au début du projet on peut distinguer deux thèmes principaux. Le premier concerne la partie mécanique et électrique qui nous donne une vision globale au robot et le deuxième est relative à la partie électronique qui est la plus importante pour ce projet.

f) Partie mécanique et électrique

Le robot est composé d´une structure autonome mobile qui porte les éléments nécessaires pour sa stabilité, mouvement en tous les directions ainsi que pour assurer la liaison robot-fauteuil et déplacer le fauteuil. Les composants les plus importantes sont :

Structure support

La structure fournit la stabilité de l´ensemble du robot et sa mobilité au moyen d´un ensemble de segments du profil structural dûment reliés et avec l´aide de deux roules folles. Cette structure héberge le reste des éléments nécessaires telles que la source d'alimentation, les roues, les moteurs, le calculateur… La Figure 1 montre la structure de base du robot:

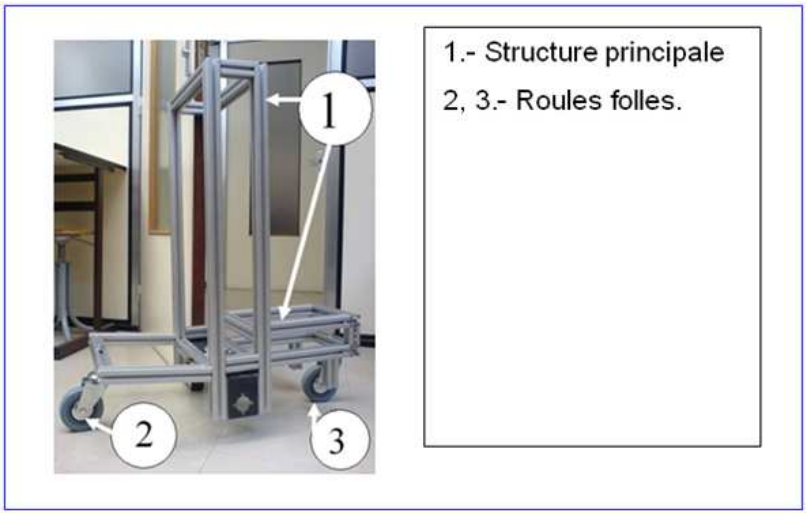

Figure 1. Structure de base du robot

Autres supports

À part la structure principale il y a autres supports dans le robot comme ceux utilisés pour la fixation des capteurs. On peut distinguer le support pour les capteurs Ultrasons (US) et Infrarouges (IR) et le support de la caméra. Ils s´affichent à la Figure 2:

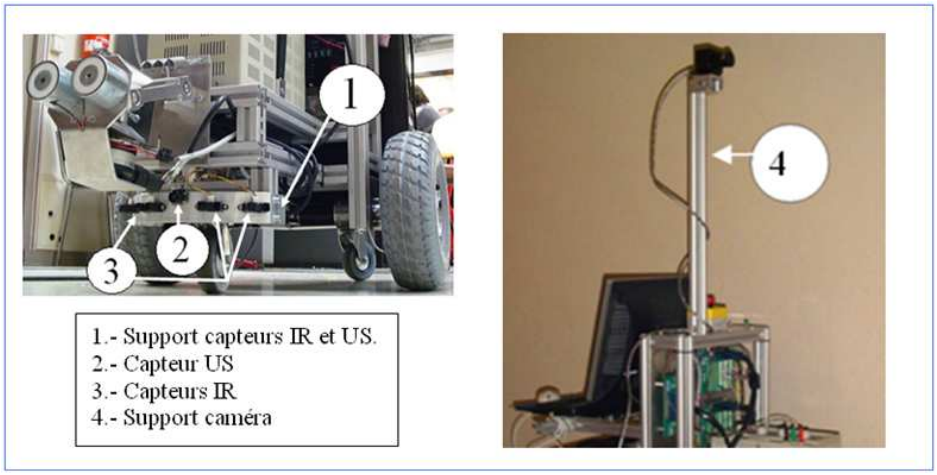

Figure 2. Support des capteurs Ultrasons et IR et de la caméra

Ventouses électromagnétiques

Les ventouses électromagnétiques sont des éléments aimantés électriquement. Il y en a deux en face du robot et elles réalisent la fonction de la liaison physique entre le robot et le fauteuil roulant. Il y a aussi un ensemble d´articulations qui transmet les efforts et assure une liaison rotoïde.

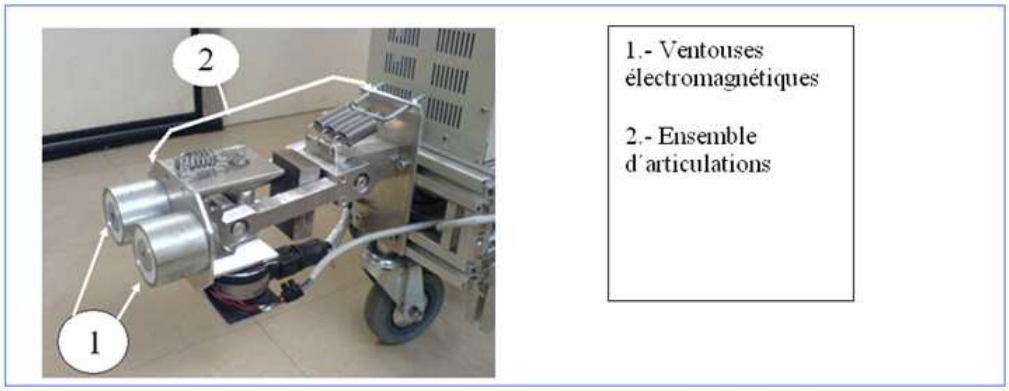

Figure 3. Ventouses et ensemble d´articulations

Roues motrices

Le robot est équipé de deux roues latérales motorisées de façon indépendantes. Ces roues permettent le mouvement dans toutes les directions. Les moteurs de chaque roue ont munies de codeur pour piloter et mesurer le mouvement ainsi que des supports pour la jonction avec le reste de la structure du robot.

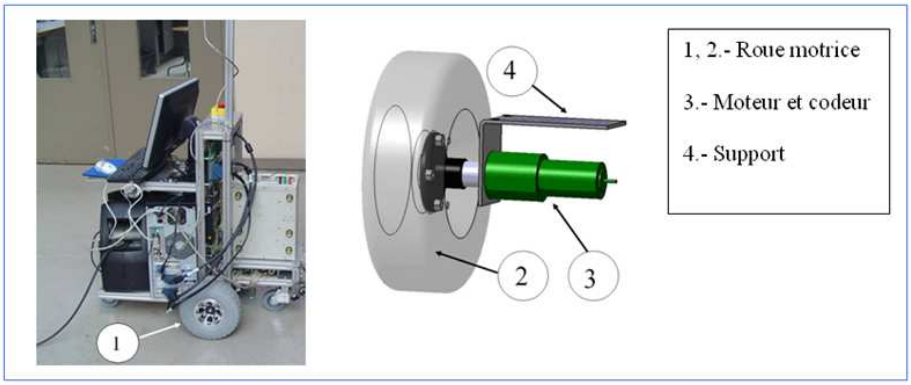

Figure 4. Roue motrice, moteur, encodeur

Source d´alimentation

La source d´alimentation est située sur la partie avant du robot et elle fournit l´énergie nécessaire à tous les composants électroniques sauf le calculateur qui a sa propre source d´alimentation. On peut la voir à la Figure 5:

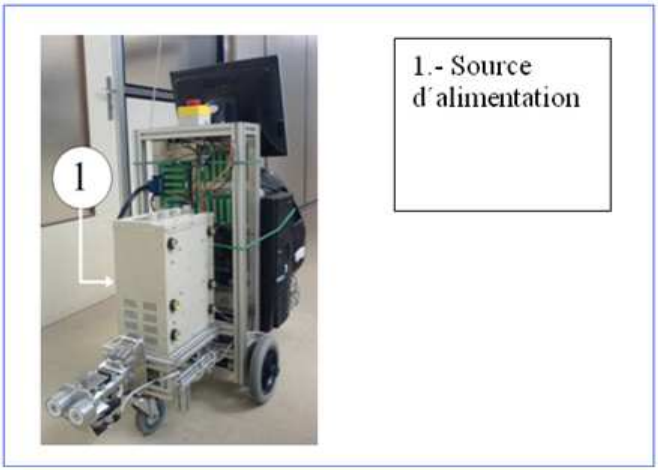

Figure 5. Source d´alimentation

Attache fauteuil

Bien qu´elle ne soit pas un composant du robot elle est aussi importante que les autres puisque elle assure la jonction robot-fauteuil et la transmission d´efforts. Elle est située derrière le fauteuil et elle doit pouvoir être placée dans no importe quel fauteuil roulant conventionnel. Elle ne doit pas provoquer une importante modification dans l´ancien fauteuil. Dans la Figure 6 on peut voir comment elle est fixée sur le fauteuil:

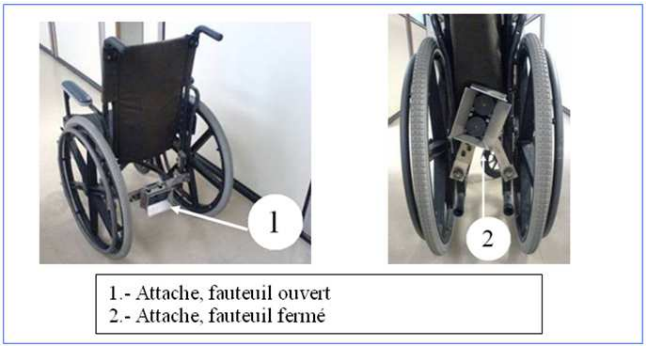

Figure 6. Attache fauteuil

g) Partie électronique

De même que l´on a décrit les composants mécaniques, on va maintenant expliquer superficiellement les composants électroniques. Ces composants sont très importants dans le développement du robot parce qu'ils nous permettent du fournir au robot de toute l`intelligence ainsi que de faire la programmation et donc la trajectographie qui est le but principal de ce projet.

Calculateur

Le calculateur est le cerveau du robot. Il est composé d´un ordinateur PC conventionnel qui facilitera l´acquisition des données, la commande des actionneurs, le travail de programmation, etc. Les logiciels utilisés pour contrôler toutes les fonctionnalités du robot sont le système opérant ˝Windows XP˝ et l´outil ˝Visual Studio 2005˝ avec l´utilisation du langage C++ et en outre les bibliothèques des cartes électroniques installées afin d'acquérir, de traiter et d'émettre des signaux électriques analogiques et numérique. Ces cartes analogiques sont reliées au robot grâce à une connectique.

Cartes électroniques

Le calculateur est complété en ajoutant de trois cartes électroniques du fabriquant ˝National Instruments˝ (NI) et qui sont nécessaires pour la communication avec les actionneurs et capteurs du robot. Les cartes ont des fonctionnalités différentes et sont affichées dans le Tableau 1:

CARTES	6221	6602.1	6602.2
Modèle	PCI 6221	PCI 6602	PCI 6602
Entrées Analogiques	16	—	—
Sorties Analogiques	2	—	—
E/S Numériques	24	40	40
Timers	—	8	8
Liaison Série/Par.	—	✓	✓
Ports USB	—	✓	✓
Liaisons	Moteurs Codeurs Roues Capteurs IR	Télécommande Encodeur Ventouses	Capteur US
E/S Libres	8A / 24N	23N	39N

Tableau 1. Tableau des cartes NI

Bornier analogique

Le bornier analogique est l'interface entre la connectique du calculateur et le câbles des modules analogiques embarqués sur le robot. Ces éléments sont fixés à une plaque métallique qui est dans le robot.

Variateurs

La fonctionnalité de les variateurs dans notre robot est de créer une interface entre les sorties des cartes NI et les moteurs. Cette fonction est nécessaire à ajouter parce que les cartes électroniques n´ont pas la puissance suffisante pour agir directement sur les moteurs des roues motrices. Il y a un variateur pour chaque moteur et ces variateurs ont aussi la fonction de transporter le signal des encodeurs des roues.

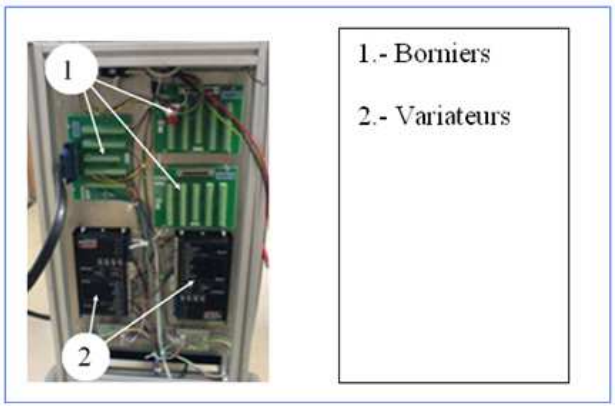

Figure 7. Plaque avec les borniers et variateurs

Capteurs

Les capteurs sont très importants parce qu'ils sont comme les yeux du robot, c´est-à-dire que ce sont les composants qui nous permettent de recevoir l´information de l´environnement comme les distances aux obstacles que le robot peut rencontrer. Cette partie est très importante pour le projet parce que, bien sûr, on a besoin de l´information fournie par les capteurs pour la programmation de la trajectoire du robot. Les capteurs sont placés dans le robot au moyen d'une ceinture installée avant du robot et qui est composé de quatre capteurs infrarouges et un capteur ultrasons. Les positions des capteurs sont affichées dans la Figure 2.

Caméra

Il y a aussi dans le robot une caméra couleur de chez XXX pour la reconnaissance de l'image et pour avoir la possibilité d'interagir avec l'environnement. Cette caméra est un capteur mais avec la différence que elle n'est pas connectée aux cartes NI mais directement au port série du calculateur.

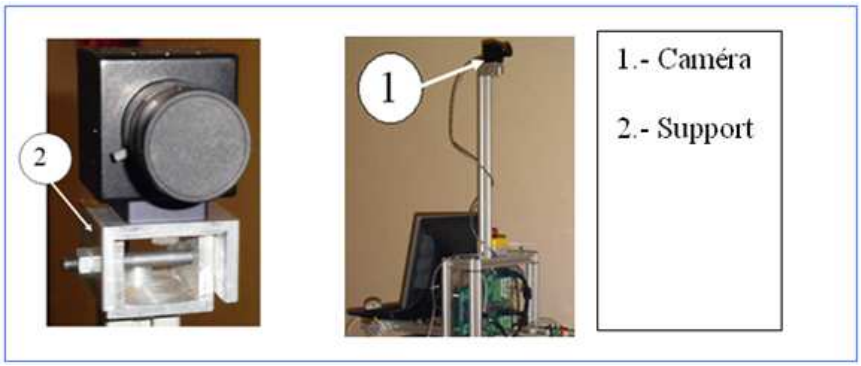

Figure 8. Caméra

Codeur angulaire

Dessous les ventouses électromagnétiques il y un codeur angulaire qui mesure la orientation (ou la rotation) du robot par rapport au fauteuil roulant. Cette information est très importante quand on programme les équations de la trajectoire du robot avec le fauteuil.

Figure 9. Codeur Angulaire

Coup de poing (arrêt d´urgence)

Dans touts les projets expérimentaux, il est nécessaire de pouvoir arrêter rapidement le système en marche si quelque chose va mal. Pour cette fonction, il y a dans le robot le ˝coup du poing˝ (arrêt d´urgence) qui coupe l´alimentation commune aux variateurs.

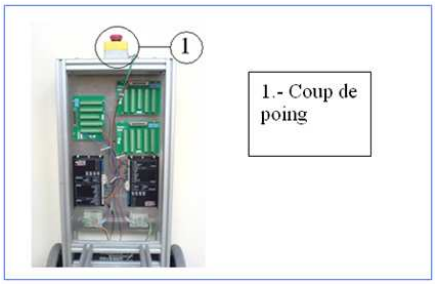

Figure 10. Coup de poing

Télécommande

Le robot dispose d'une télécommande pour la commande à distance. Ainsi les personnes handicapées pourront elles mêmes piloter le fauteuil. Elle fonctionne au moyen d'une application déjà développée sous Visual C++ et des cartes NI.

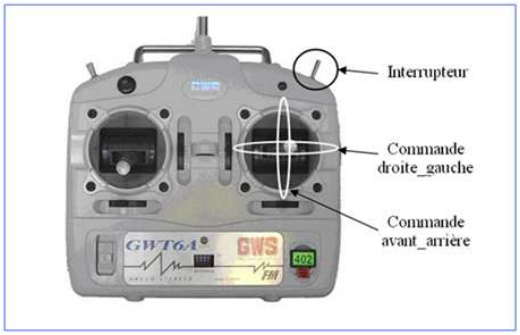

Figure 11. Télécommande

Tous ces renseignements ont été obtenus à partir des rapports des projets des années précédentes. Pour plus d´information, on peut consulter les rapports [10], [11], [12] et [13] référencés en bibliographie.

5. PROGRAMMATION

Dans ce chapitre on va résumer tout ce qui concerne à la programmation d´un point de vue général, théorique et pratique mais en se limitant à ce qui est utile pour ce projet.

D´abord, on commence par expliquer les concepts généraux de la programmation dans Visual C++. Ensuite, on continue avec les renseignements sur l´utilisation du logiciel du développement Visual Studio 2005 et les fonctionnalités additionnelles fournies par la bibliothèque National Instrument.

a) Visual C++

Structure d´un programme Windows

Quand on développe un programme dans Visual C++ on opère en 2 étapes:

Générer la code automatiquement.
Modifier et enrichir le code en fonction de nos besoins.

Sous sa forme la plus réduite, un programme écrit exclusivement à l´aide de l´API Windows comporte deux éléments fondamentaux:

Une fonction `WinMain()` appelée au début de l´exécution du programme et contenant son initialisation de base,
Procédure de fenêtre `WindowProc()` ou `WndProc()` : appelée par le système d´exploitation chaque fois qu´un message doit être transmis à la fenêtre de notre application.

Les tâches de chacune de ces deux fonctions développent sont:

WindMain() :

Operations d´initialisation
Configure les fenêtres de l´interface principal

WindowProc() :

Gère les messages (par placés dans la file de messages)
Vous codez réponse à chaque message Windows
Gère toutes les communications avec l´utilisateur.

Ces deux fonctions composent un programme complet, mais elles ne sont pas directement liées. En effet, `WinMain()` n´appelle pas `WindowProc().` C´est Windows qui appelle cette fonction. En fait, c´est également Windows qui appelle `WinMain(),` comme le montre l´illustration suivante:

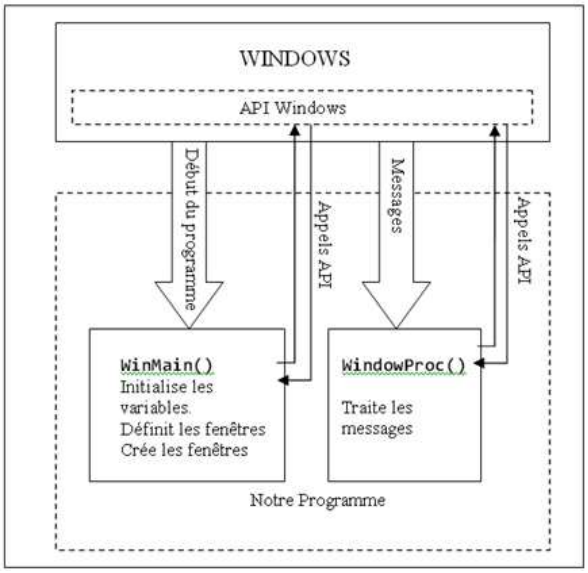

Figure 12. Liens entre notre programme et Windows

On appelle API Windows à l´interface de programmation d´applications Windows. Chaque programme Windows utilise l´API Windows et MFC facilitent considérablement l´utilisation de l´API.

Maintenant, on va expliquer de façon générale le MFC.

MFC (Microsoft Foundation Classes)

Les MFC constituent un ensemble de classes prédéfinies, autour desquelles s´articule la programmation Windows avec Visual C++. Elles composent une structure complète de développement d´applications, où il nous suffit d´effectuer les opérations de personnalisation dont nous avons besoin pour adapter les programmes à nos exigences.

La programmation d´un programme Windows entraîne la création et utilisation d´objets des MFC(ou classes dérivées des MFC). Nous créons nos classes à partir des MFC. Les classes dérivées héritent naturellement de tous les membres de leurs classes de base. Elles se chargent d´effectuer pratiquement toutes les tâches nécessaires au fonctionnement d´une application Windows.

Ensuite, pour doter nos programmes des fonctionnalités spécifiques on doit ajouter fonctions membres et les membres données qui nous permettent de personnaliser les classes.

Il est intéressant d´examiner les relations réciproques entre les classes fondamentales d´une application SDI (Single Document Interface) et les MFC, comme le montre la Figure 13:

Figure 13 illustre les quatre classes élémentaires qui se trouvent dans pratiquement toutes nos applications Windows :

La classe d´application, CMyApp

La classe de fenêtre cadre, CMyWnd

La classe de vue, CMyView, qui détermine l´affichage des données contenues dans CMyDoc dans la zone client d´une fenêtre créée par un objet CMyWnd.

La classe de document, CMyDoc, qui détermine un document devant contenir les données d'application.

Convention: Toutes les classes des MFC portent des noms qui commencent par C (CDocument, CView,…) et les membres données d´une classe MFC commencent par le préfixe m_.

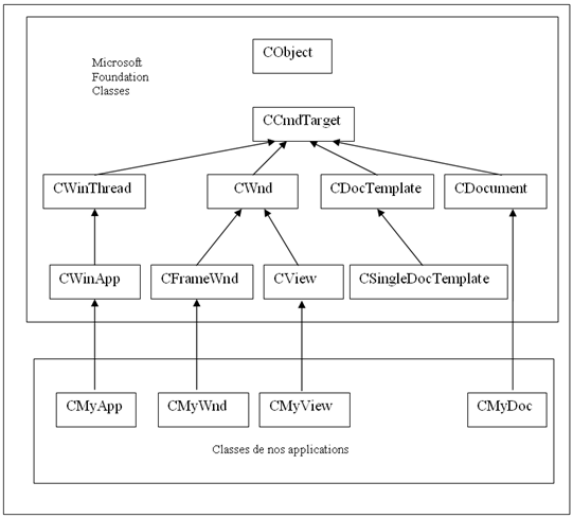

Figure 13. Relations entre les classes fondamentales d´une application SDI et les MFC

Notre programme ne contient aucune définition des classes de base, il mentionne seulement les membres hérités d´une classe dérivée de la classe de base directe, ainsi que de chacune des classes de base indirectes de la hiérarchie des MFC.

A la Figure 13, chaque flèche part de la classe de dérivée et pointe vers la classe de base.

Les Classes

Une classe désigne un type de données défini par l´utilisateur. Elle peut contenir des données qui sont soit des variables des types élémentaires du C++, soit d´autres types de données définies par l´utilisateur. Elle peut également contenir des fonctions opérant sur ses objets en accédant à leurs éléments de données. Ainsi, une classe combine la définition des données élémentaires constituant un objet et les moyens permettant de manipuler les données appartenant à des objets de la classe.

Les données membres désignent les données d´une classe et les fonctions membres (aussi connues comme méthodes) désignent les fonctions d´une classe.

La définition d´une classe équivaut à la création d´un type de données. Ce ne sont pas les données qui sont définies, mais la signification du nom de la classe, c´est-à-dire le contenu d´un objet de la classe et les opérations pouvant être effectuées sur l´objet.

Le concept Document/Vue

Lorsque nous écrivons des applications au moyen de MFC, nous dotons notre programme d´une structure spécifique, ses données étant stockées et traitées d´une certaine manière.

La structure d´un programme MFC intègre deux entités orientées application, un document et une vue.

Dans la suite, on va examiner leur signification et leur mode d´utilisation.

Le document

Un document désigne l´ensemble des données de notre application, avec lequel l´utilisateur communique. Le terme "document" est un moyen pratique de qualifier les données de notre programme, traitées comme une unité.

Il est représenté par un objet d´une classe de document, dérivée de la classe `CDocument` de la bibliothèque MFC. La classe dérivée ajoute les membres donnés nécessaires au stockage des éléments propres à l´application, ainsi que les fonctions membres prenant en charge le traitement de ces données.

Interfaces de document

Nous avons la possibilité d´indiquer que notre programme traite un seul document ou plusieurs à la fois. La bibliothèque MFC prend en charge l´interface de document unique, SDI (*Single Document Interface*), pour les programmes devant ouvrir un seul document à la fois. Ce type de programme est connu sous le nom d´application SDI.

Pour les programmes devant ouvrir plusieurs documents à la fois, il faut avoir recours à l´interface de documents multiples, MDI (*Multiple Document Interface*). MDI permet non seulement l´ouverture de plusieurs documents de même type, mais également la gestion simultanée de documents de différents types.

La vue

La « vue » concerne toujours un objet document particulier. Elle propose un mécanisme permettant d´afficher tout ou partie des données stockées dans un document. Elle définit le mode d´affichage des données dans une fenêtre, et le mode de communication avec l´utilisateur. Pour définir notre classe vue on le fait à partir de la classe MFC `CView`. Il est bon faire une distinction entre la vue et la fenêtre où elle s´affiche, appelée fenêtre cadre (*frame window*). Une vue (fenêtre vue) s´affiche dans la zone de la fenêtre cadre.

Plusieurs objets « vue » peuvent être associés à un objet document. Chaque objet « vue » peut proposer une présentation différente des mêmes données du document.

A la Figure 14 est représentée la relation entre un document, une vue et une fenêtre cadre:

Dans la Figure 14, la vue affiche seulement une partie des données contenues dans le document, alors qu´elle peut en afficher la totalité si nécessaire.

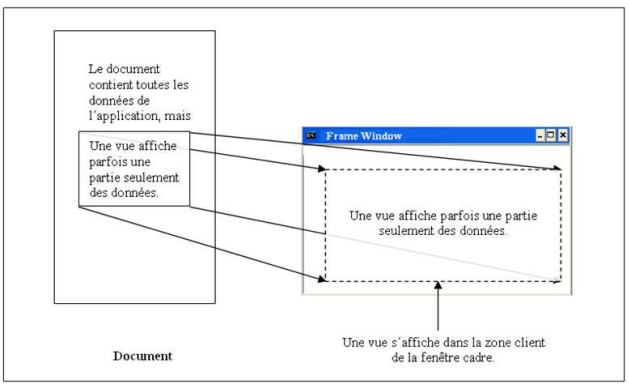

Figure 14. Relation entre un document, une vue et une fenêtre cadre

Liaison d´un document avec ses vues

Les MFC intègrent un mécanisme permettant de lier un document avec ses vues, et chaque fenêtre cadre avec une vue active. La coordination entre un document, une vue et une fenêtre cadre est assurée par une autre classe d´objets MFC, appelés modèles de document.

Un modèle de document (*document template*) gère les documents de notre programme, ainsi que les fenêtres et les vues associées à chacun d´eux. Il existe un modèle pour chaque type de document de notre programme. Si vous possédez plusieurs documents d´un même type, un seul modèle suffit pour les gérer. Le modèle de document est créé par l´application et c'est un objet fondamental de tout programme MFC.

La bibliothèque MFC contient deux classes qui permettent définir les modèles de document :

Pour les applications SDI: CSingleDocTemplate (contiennent un seul document, et généralement une seule vue).

Pour les applications MDI: CMultiDocTemplate (peuvent contenir plusieurs documents actifs simultanément).

Par conséquent, à la fin les deux classes travaillent ensemble. C´est pour cela que l'on fait l'intégration document-vue, ce qui permet modifier les données représentées à l'écran lorsqu'elles changent dans le document et vice versa.

Il y a trois différentes procédures pour faire l´intégration document-vue:

Quand le document demande l´actualisation des vues.
Quand la vue demande la nouvelle valeur de la donnée au document.
Quand on établit un protocole spécifique document-vue.

Bien que le procédé le plus recommandé est le premier, dans notre projet on va utiliser le deuxième qui est le plus adapté aux besoins. On peut voir graphiquement cette intégration dans le schème suivante:

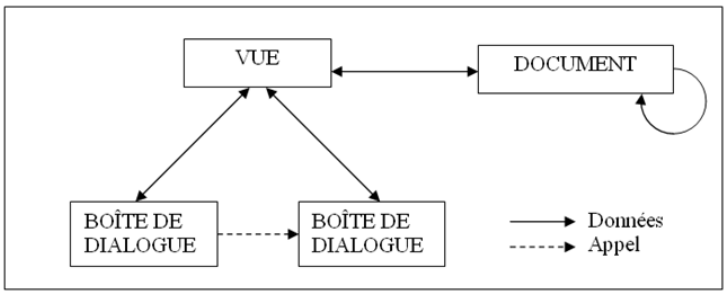

Figure 15. Intégration Document-Vue

Notre choix

L'architecture choisie parmi les différentes architectures existantes dans l´outil de programmation et développement Visual Studio 2005 et le langage Visual C++ est l´architecture Document-Vue.

La raison principale de ce choix est que c´est un choix fonctionnel et pratique. L´alternative la plus proche serait l´architecture Dialog-Based qui ne permet pas trop de complexité et qui manque de certaines

fonctionnalités, de barres et de boîtes d´outils, etc. Et en plus, comme on l'a déjà dit, elle organise et structure toute la programmation selon deux classes, le document et la vue.

D'autres justifications valides sont:

L´héritage des projets antérieurs qui ont utilisé cette architecture et le plus sensé est de la suivre.

La recommandation du fabricant National Instruments

On peut trouver beaucoup de bibliographie sur ce sujet, même sur internet. Pour obtenir des informations complémentaires, on conseille la référence 0.

b) Visual Studio 2005

Visual Studio 2005 est le logiciel qui est utilisé dans le robot et qui sert au développement des applications. Pour cette raison on va expliquer quelques notions de base sur son utilisation ainsi que sur l´adaptation de l´architecture Document-Vue.

Créer un nouveau project

La première chose que on doit faire est démarrer le menu de création d´un projet : ***Fichier → Nouveau → Projet***. On ouvrira une fenêtre appelé ***Nouveau Projet*** comme on peut voir à la figure 16 :

La figure 16 montre comment, pour créer une application exécutable dans le calculateur du robot, on doit choisir dans **Types de projets** l'option ***MFC*** et dans **Modèles Visual Studio Installés** l´option ***Application MFC***. Après, il faut sélectionner un nom et un emplacement et appuyer ***OK.*** Grâce aux MFC le procédé de programmation deviendra beaucoup plus facile.

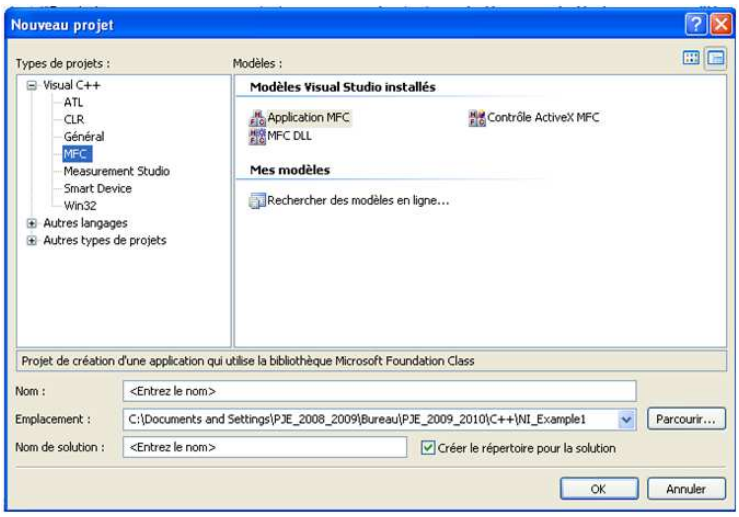

Figure 16. Fenêtre Nouveau Projet

Après la validation du nom, il apparaît une fenêtre ***Assistant Application MFC*** qui sert d'aide pour la création du nouveau projet. A gauche s´affichent les différentes étapes que l'on doit suivre et à droite les différents choix que l'on peut choisir pour la configuration base du projet dans chaque étape.

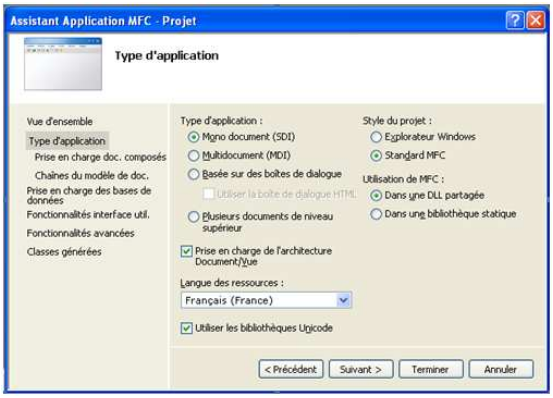

Figure 17. Fenêtre Type d´application dans l´Assistant Application MFC

On peut voir à la Figure 17 ces choix que l'on peut changer pour adapter les caractéristiques du projet à l´application souhaitée. L´élément le plus important est le choix ***mono-document*** **(SDI).** Les autres choix de cette étape étant validés. Il reste à cliquer ***Suivant***.

Ensuite, on clique ***Suivant*** en laissant tous les autres choix par défaut jusqu´a ce qu'on arrive à l'étape ***Classes générées*** et on choisit la classe de base ***CFormView*** pour la classe vue comme on peut le voir à la Figure 17:

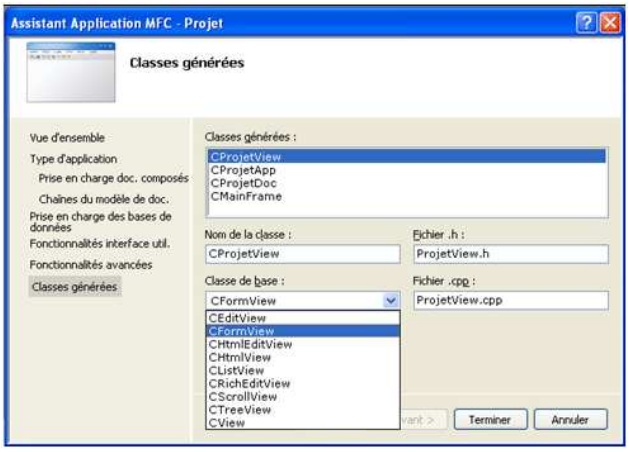

Figure 18. Fenêtre Classes Générés dans l´Assistant Application MFC

Comme on a choisi *CFormView* l´application que l'on obtient possèdera une boîte de dialogue dans la fenêtre cadre principale où l´on pourra placer tous les contrôles nécessaires pour la vue. Dans cette fenêtre, on peut aussi voir les classes qui vont être générées et modifier les noms proposés. On ne change pas les noms.

Appuyer ***Terminer*** pour obtenir le résultat de notre nouveau projet qui est affiché dans la figure:

Expliquons la Figure 19 puis que c'est la fenêtre principale de travail de *Visual Studio 2005*. Cette fenêtre correspond à la fenêtre de travail créée à l´aide de l´Assistant Application MFC pour un projet avec l´architecture Document-vue. D´abord on trouve la barre de menu classique qui existe dans les applications *Microsoft* (1) où on peut apprécier les menus fichier, édition, fenêtre et aussi *Measurement Studio* puisque le logiciel pour l'intégration des cartes NI et le Visual C++ est déjà installé. Juste au dessous on trouve la barre d´outils (2) avec des éléments d´édition et que l'on peut modifier.

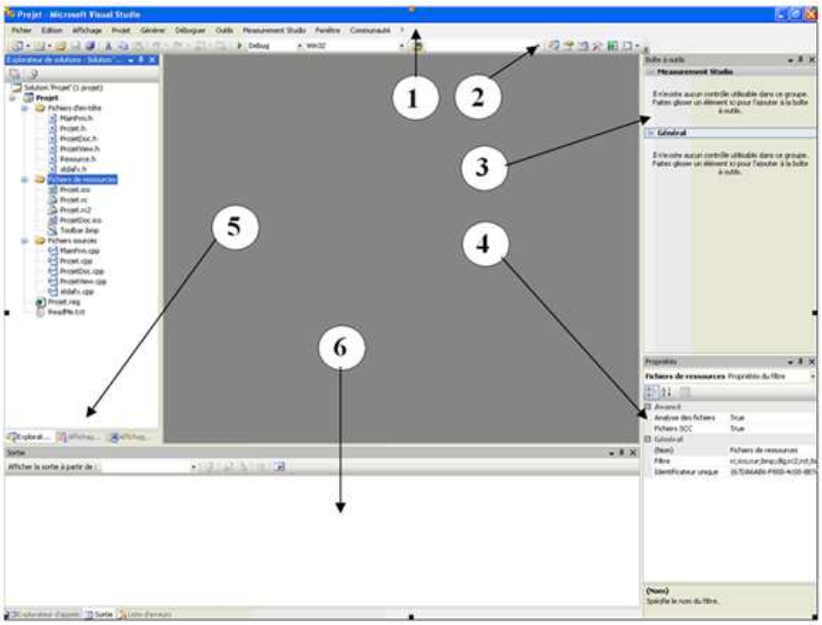

Figure 19. Fenêtre Principale projet Visual Studio 2005

À droite on voit la boîte à outils (3), qui est utilisée principalement lors du développement des ressources car c´est là que les composants graphiques s'affichent. On a aussi la fenêtre des propriétés (4), qui est une des plus importantes lors de développement de la vue. Chaque fois qu´on sélectionne une ressource toutes ses propriétés apparaissent avec leurs identificateurs (IDx_NOM). De même, si l'on peut joindre un événement de contrôle, on le peut faire en appuyant sur l´icône avec une foudre.

À gauche on trouve l´explorateur de solutions, l´affichage de classes et l´affichage de ressources (5). Ce sont différents moyens d´afficher les informations contenues dans le dossier du projet et proposées par Visual C++:

Onglet et Vue	Affichage du projet
Explorateur de solutions	Par nom de fichiers
Affichage de classes	Par noms de classes et de fonctions membres, plus les entités globales du programme
Affichage des ressources	Par type de ressources

Tableau 2. Types d´affichage les informations du projet

On utilise le premier pour ouvrir les différents fichiers inclus dans le projet. Avec le deuxième, on peut voir le projet structuré dans ses classes et les fonctions membres et on peut atteindre leur déclaration et leur définition. Le dernier nous permet d'ouvrir les ressources qui appartiennent au projet et permet leur modification.

Enfin, tout en bas, on trouve l´affichage des sorties (6). Cet espace est réservé pour l´affichage de tous les renseignements apportés par rapport aux procédés réalisés sur le projet. L'utilisation la plus importante et la plus courante est celle de l'analyse du résultat de la génération puisque toutes les erreurs y apparaîtront et, en faisant un double click ou en appuyant F1 sur ces erreurs, on se dirigera à la ligne où ils se trouvent et F1 fournit toute l'information nécessaire pour corriger cette ligne.

Concevoir une interface d´usager

Maintenant on va expliquer comment on peut manipuler et changer l´interface de l´application pour lui donner la fonctionnalité souhaitée. On peut faire tout cela grâce à l'ajout et la modification des ressources de plusieurs types comme par exemple le menu avec ses options, un bouton sur la barre d´outils, un bouton animé sur le dialogue, etc.

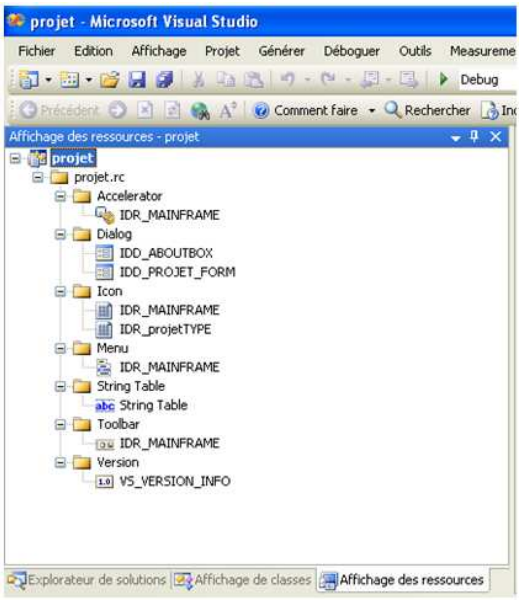

Figure 20. Affichage des ressources

A la Figure 20 on peut voir tous les genres de ressources classiquement disponibles dans une architecture document-vue. Si on ouvre les dossiers existants dans la fenêtre d'affichage des ressources on trouve les accélérateurs, les dialogues, les icônes, le menu, la table des chaînes, les barres d'outils et la fenêtre de la version. Toutes ces ressources sont les composantes du fichier NOM.rc inclus dans le projet. Ensuite, on expliquera les ressources les plus intéressants pour le développement d'une application.

La barre de menu

La barre de menu inclue les fonctionnalités typiques des applications comme le menu fichier, édition, affichage et aide. On peut modifier les applications existantes dans cette ressource ou ajouter autres nouvelles.

Par exemple, dans la Figure 21 on a ajouté un ***Nouveau Menu*** et une ***Nouvelle Option.*** Tout cela peut être fait tout simplement en tapant les noms dans les boîtes où on indique ***Tapez ici***. Pour rappeler les raccourcis on peut taper & avant la lettre que l´on veut souligner. En double cliquant dans le nom de la nouvelle option ajoutée (ou une autre option que l'on voudrait voir) ou avec le bouton droit de la souris on peut accéder aux propriétés qui affichent tous les paramètres de l'option choisie. Le plus important est l´identificateur, qui peut être modifié et à travers lequel on donne la fonctionnalité à l'option.

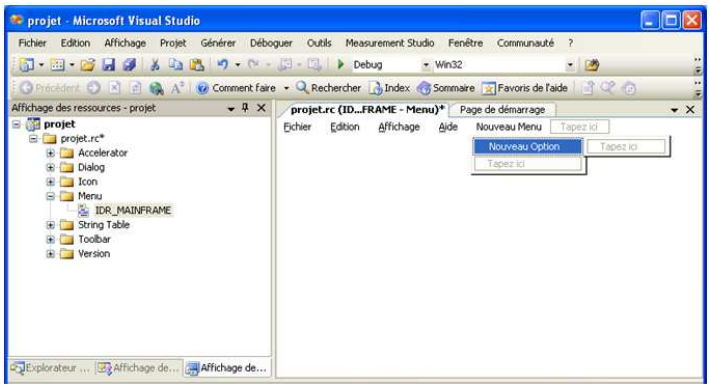

Figure 21. Modification Barre de Menu

La barre d´outils

La barre d´outils est très semblable à la barre de menu. La différence est que dans la barre de menu il y a un menu de mots et dans la barre d´outils il y a boutons graphiques dessinés par l´utilisateur.

Ce design est fait à l´aide d`une boîte à outils spéciale pour dessiner icônes. On peut voir dans la Figure 22 comment on a dessiné une face souriante pour un nouveau bouton. Lors de l´édition d´un icône, dans la boîte de propriétés (que on peut accéder commet on l´a expliqué avec le barre de menu), on voit parmi d´autres, l´identificateur du bouton. Si on choisit le même identificateur qu´une option de la barre de menu on va obtenir la même fonctionnalité pour les deux boutons.

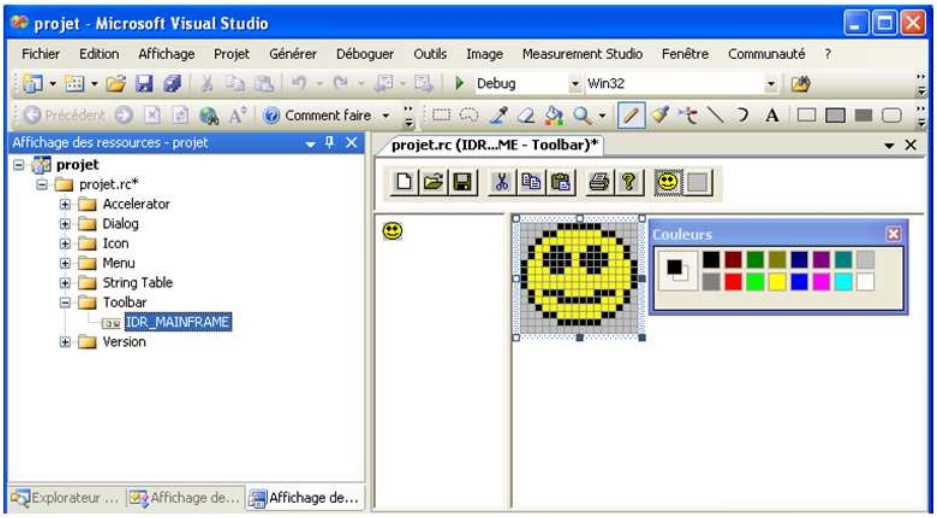

Figure 22. Modification Barre d´outils

La boîte de dialogue principale

Expliquons ici la conception d'une boîte de dialogue principale, qui sera encadrée dans la fenêtre cadre de l'application.

Dans la boîte de dialogue on trouve les composants les plus importants de l´application.

Pour son édition on utilise les éléments spécifiques qui se trouvent dans la boîte à outils (Partie droite de la Figure 23). Parmi ces composants il y en a quelques uns appartenant à *Measurement Studio* de NI, puisque le logiciel d´intégration a été déjà installé.

Dans la Figure 23 on voit un exemple d'une boîte de dialogue avec plusieurs composants qui permettant plusieurs possibilités d'édition:

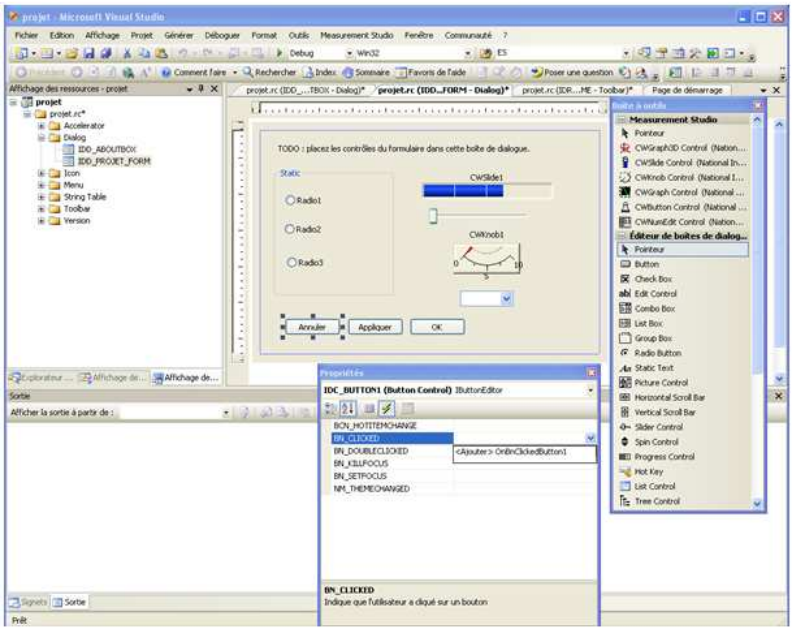

Figure 23. Boîte de dialogue principale

On peut accéder à la boîte des propriétés de la même façon que celle qui a été expliquée dans les sections précédentes. On trouve l´identificateur de l'élément choisi que l'on peut modifier en fonction de nos besoins. Il y a aussi le gestionnaire d´événements (représenté par une foudre) que l'on peut ajouter pour gérer l'élément. Pour cela, il suffit d'appuyer sur la foudre, choisir l'événement souhaité et appuyer à droite sur ***<Ajouter> OnXXXXX***. La Figure 23 présente comment s´ajoute le gestionnaire d´événement pour le bouton ***Annuler***. Ensuite, *Visual Studio* va directement vers le segment de la classe vue où l'on peut taper le code nécessaire pour faire fonctionner le composant (fichier *NOMView.cpp. ProjetView.cpp en notre cas*). Le code prédéfini peut être vu ci-dessous:

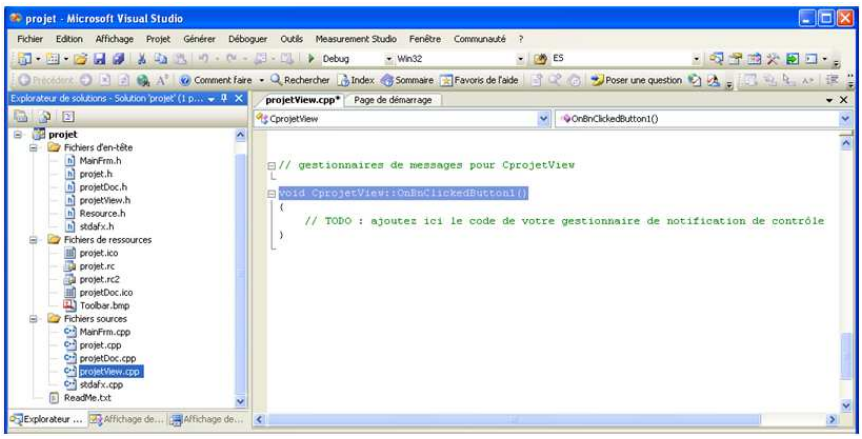

Figure 24. Code prédéfini. Gestionnaire d´événements

Les accélérateurs et la table des chaînes

La liste des accélérateurs comprend les identificateurs des ressources associés à une combinaison des touches du clavier : cela permet de raccourcir le temps d'accès à une option ou à une fonctionnalité.

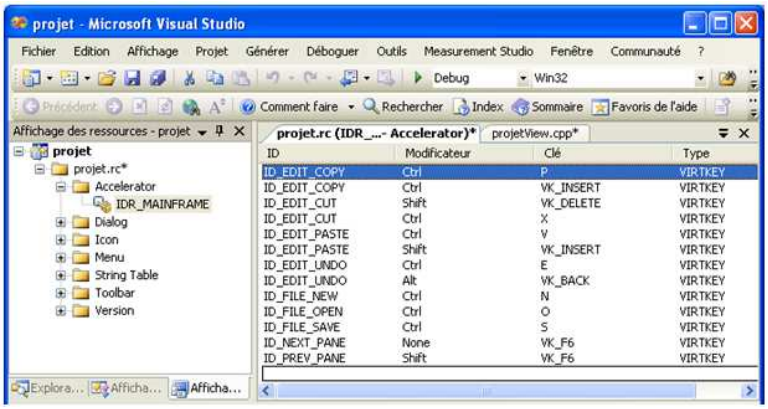

Figure 25. Accélérateurs

La table des chaînes montre les identificateurs des ressources, leurs valeurs numériques et leurs légendes (l'étiquète dans le menu joint avec l'explication qui apparaît lorsque la souris devient immobile).

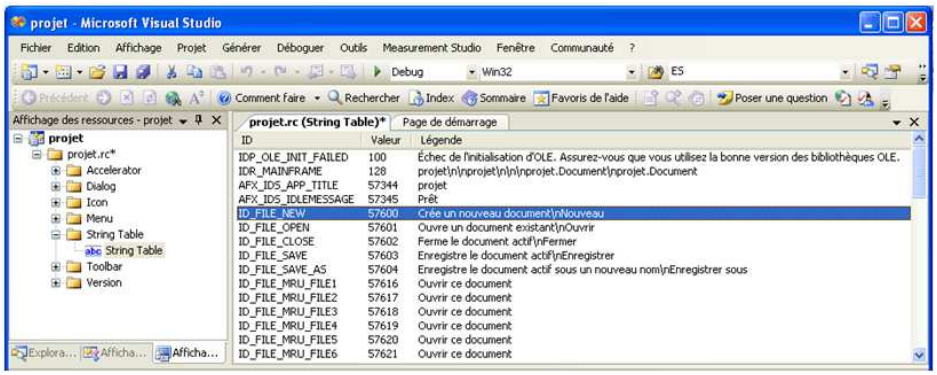

Figure 26. Table de chaînes

Pour obtenir plus d`information, on peut recourir aux références 0 et 0. Il y a aussi beaucoup d´informations complémentaires sur internet.

Architecture Document/Vue dans Visual Studio

On a déjà signalé que l`architecture *Document-Vue* est l´architecture dans laquelle on va développer ce projet et aussi on a expliqué quelques unes de ses avantages et les motifs de ce choix.

D´autre part, on a indiqué ce qu´il faut faire pour créer et modifier une application dans Visual Studio 2005 avec cette architecture. Maintenant, on va analyser les fichiers que ce logiciel génère et ses liaisons.

Dans l'architecture *Document-Vue* il y a deux classes fondamentales. Ce sont les classes document et vue. Pour gérer ce sujet, Visual Studio crée plusieurs fichiers d´extensions différentes telles que ***.ccp*** (code des fonctions), ***.h*** (déclarations), ***.rc*** (ressources), etc. Mais les fichiers qui définissent cette organisation sont ***NOMDoc.cpp/.h*** et ***NOMView.cpp/.h***. Chaque couple représente une classe composant le document ou la vue. De plus, il y d'autres fichiers tels que ***MainFrm.cpp/.h*** qui gèrent la fenêtre cadre de l'application ***NOM.cpp/.h***.

Chaque fichier est composé de :

le fichier ***NOMDoc.cpp*** qui contient le code des fonctions dédiées au traitement des données tandis que le fichier ***NOMDoc.h*** contient les déclarations de ces fonctions ainsi que des variables ou données à utiliser (incluses les classes de données).

le fichier ***NOMView.cpp*** possède les fonctions membres et gestionnaires nécessaires pour gérer l'interface de l'utilisateur et le fichier ***NOMView.h*** aura toutes les déclarations des variables et classes utilisées dans ce but; ainsi que tout ce qui concerne les ressources développées.

La Figure 27 montre les liaisons qui permettent la bonne marche de cette technique:

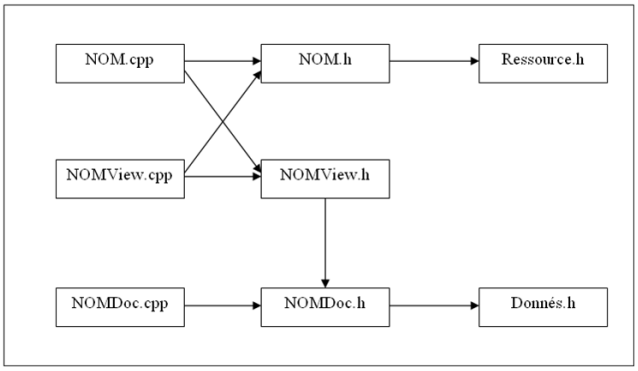

Figure 27. Rapports entre les fichiers

Pour toutes ces liaisons et le bon fonctionnement il y a un rapport administré à travers du mot réservé du langage C ***#include <NOM.EXT>.*** On peut dire que chaque fichier .cpp est lié à un fichier .h. De plus les fichiers généraux de l'application sont reliés aussi bien aux ressources de la vue et du document qui contient indirectement toutes les déclarations.

Pour terminer, précisons que le fichier *Données.h* et le fichier *Données.cpp* sont associés et qu'ils définiront ensemble une classe de données générique.

c) Measurement Studio de National Instruments

Présentons comment utiliser quelques des fonctionnalités fournies par le logiciel de NI. On a déjà parlé des cartes NI et leurs fonctions dans le projet. On a aussi commenté la nécessité d´installer le logiciel *Measurement Studio* de *National Instruments* pour intégrer ces cartes et leur commande dans Visual Studio et dans l´application.

La Figure 28 est à un exemple de programmation avec les composants de *Measurement Studio* développés en l´annexe a). On peut remarquer dans cette figure l´édition des ressources d´un dialogue avec composants du *Measurement Studio*:

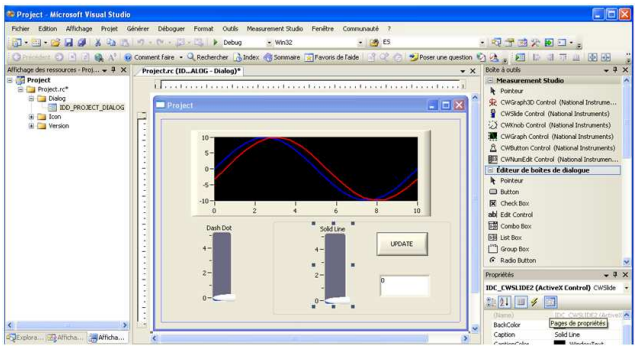

Figure 28. Edition Dialog avec composants Measurement Studio

Dans la Figure 28 on trouve la boîte de dialogue qui a un graphique avec deux différents signaux sinusoïdaux qui varient en amplitude et en couleur à l´aide des contrôles graphiques et numériques. Dans la partie droite de la figure on peut voir la boîte à outils avec les ressources apportées par *Measurement Studio* et celles apportées par Visual Studio. Si on accède à la fenêtre de propriétés (comment on l'a déjà expliqué) on peut modifier l´identificateur, ajouter les gestionnaires d´événements, etc. Mais, maintenant, il y a une nouvelle fonctionnalité. Si on appuie sur le bouton ***pages des propriétés*** (le bouton le plus à droite de la boîte de propriétés que l'on peut voir dans la Figure 28) on verra une boîte de dialogue qui permet de changer les caractéristiques et l´apparence des composants. On va montrer quelques exemples:

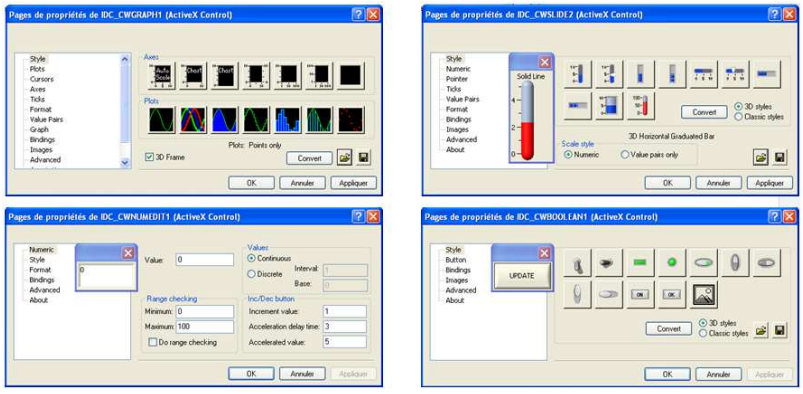

Figure 29. Pages des propriétés d'un bouton d'un Graph Control, d'un Slide Control, d'un NumEdit Control et d'un Button Control

La classe CNiControl, dérivée de CWnd gère toutes ces fonctionnalités. Cette classe contient aussi les dérivées CNiButton, CNiGraph, CNiGraph3D, CNiKnob, CNiNumEdit et CNiSlide. En modifiant les paramètres sur ces boîtes et directement sur l'édition des ressources on aura une grande flexibilité pour la conception de l'interface.

6. PROGRAMMATION DE LA TRAJECTOIRE POUR EVITER OBSTACLES

a) Programme existant. Robot_HAMMI v2.2

Tout d´abord il faut commencer par bien connaitre ça qui a été développé sur le robot pendant tous les années précédentes.

Le programme qui commande le robot a été amélioré successivement dans toutes ces années en créant les différentes versions. C´est pour cela que l`on a analysé la plus performante des versions du logiciel qui gère le robot. La version *Robot_HAMMI v2.2.*

Robot_HAMMI v2.2

Ce programme a par but la commande du robot HAMMI pour effectuer une trajectoire droite entre deux points et puis pour décrire un arc de cercle. Il inclut la loi de commande, la communication avec les moteurs et les codeurs, le traitement et l´estimation de données acquises par les capteurs ainsi que les fichiers de base (document, vue, déclarations, etc.)

C´est un programme très grand qui se compose de 17 classes, 25 fichiers d´en-tête et 21 fichiers source. On peut vérifier la taille de ce programme dans les figures suivantes:

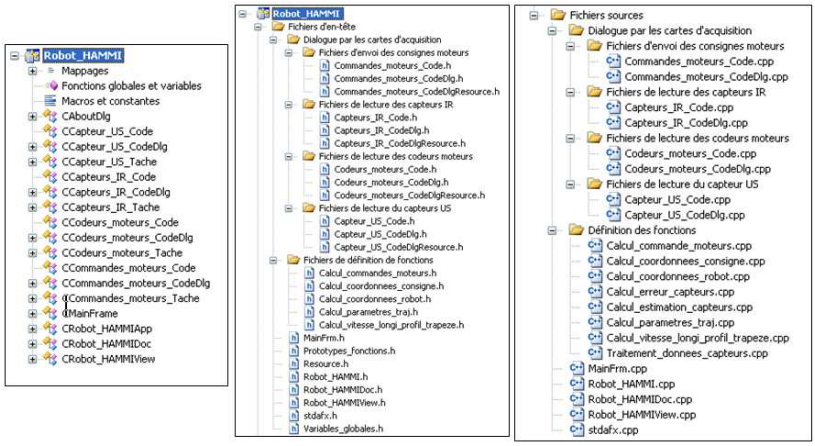

Figure 30. Affichage des classes, fichiers d´en-tête et fichiers sources

Expliquons les composants les plus importants de ce programme et ses fonctionnalités:

Comme on l´a déjà vu, la programmation selon l´architecture qu'on a suivi se concentre dans les classes CNOMDoc et CNOMView.

CRobot_HAMMIView

Cette classe est essentielle dans notre cas car elle définit la vue de l´application. Elle contient les boucles d'acquisition et de traitement des données. La déclaration de ses membres apparaît dans le fichier Robot_HAMMIView.h et la définition de ses membres apparaît dans le fichier Robot_HAMMIView.cpp.

CRobot_HAMMIDoc

Cette classe gère en fait le document de l'architecture Doc/View. Elle contient par exemple les initialisations de variables nécessaires. La déclaration de ses membres apparaît dans le fichier Robot_HAMMIDoc.h et la définition de ses membres apparaît dans le fichier Robot_HAMMIDoc.cpp.

D´autres composants très importants sont :

Prototype_fonctions.h

Fichier d'en-tête dans lequel sont déclarés les prototypes des fonctions servant à calculer les consignes de vitesse à appliquer aux moteurs pour suivre la trajectoire désirée.

Variables_globales.h

Fichier d'en tête qui définit toutes les variables globales utilisées dans le programme.

CCodeurs_moteurs_Code/CodeDlg

Ces classes permettent d'effectuer simplement l'acquisition des signaux des codeurs.

CCommandes_moteurs_Code/CodeDlg

Ces classes permettent d'effectuer simplement l'envoi des consignes aux moteurs.

Traitement_donnees_capteurs.cpp

Ce fichier réalise le filtre numérique et l´étalonnage des mesures.

Ensuite, l'interface d'usager de ce logiciel existant au début du projet avec ses parties expliquées :

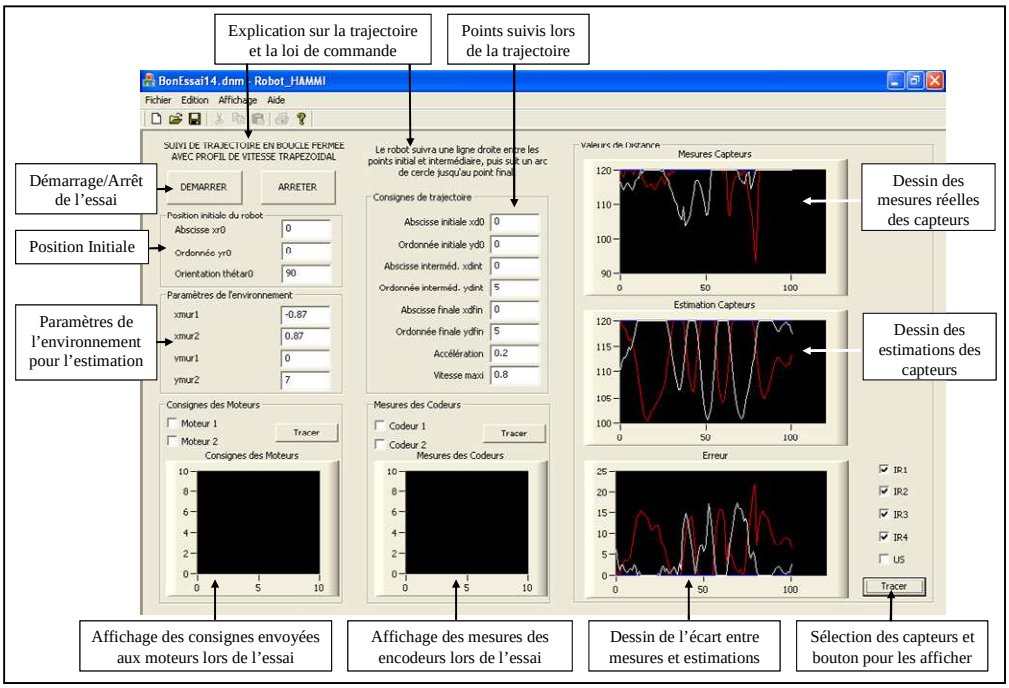

Figure 31. Interface d'usager de Robot_HAMMI v2.2

Une fois que l´on a présenté le programme existant on peut décrire ce que l´on a amélioré dans le programme.

b) Développement du nouveau programme

Dans cette section on résume tout ce qui concerne à la programmation du calculateur embarqué sur le robot. Les changements réalisés sur le programme décrit auparavant, ont permis d´atteindre une nouvelle version, le *Robot_HAMMI v3.2.*

Dans version précédente quand trouve un obstacle le robot s'arrête. Dans cette nouvelle version, on se concentre sur la trajectographie pour éviter les obstacles possibles que le robot peut se trouver et ainsi pouvoir suivre son chemin. Il y a aussi une nouvelle tâche que le programme peut réaliser. On peut enregistrer les donnees stockées dans la matrice m_donnees de notre programme dans un fichier externe pour son traitement ultérieur.

Il faut dire que tout les travaux réalises sur le robot sont faits sans tenir compte du fauteuil roulant, cela veut dire que les mouvements sont programmés pour le robot seul.

Il y a certaines caractéristiques dans l´interface de l´usager de la version v2.2 qui n´ont pas été utilisés mais on n´a rien supprimé au cas où elles seraient utiles dans les prochains projets. Par conséquent l´interface d´usager de la version v3.2 est un peu modifié pour ajouter les outils nécessaires pour l´enregistrement des donnees dans un fichier par rapport à la version v2.2.

Voici l´interface de l´usager principale de la version v3.2

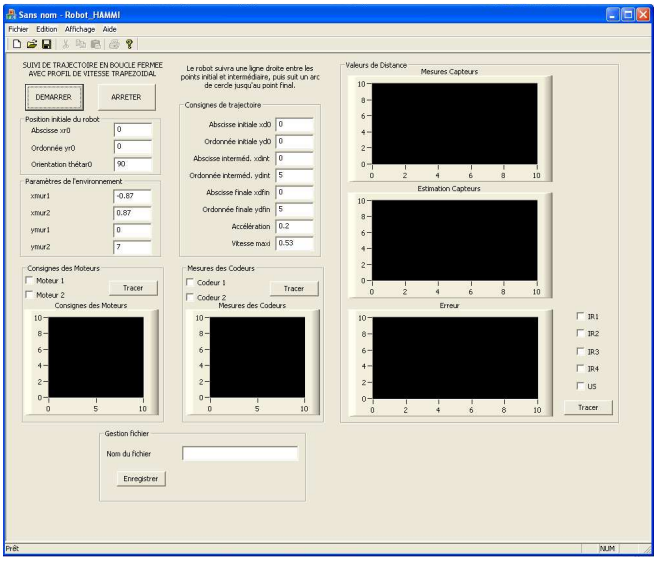

Figure 32. Interface de l'usager du Robot_HAMMI v3.2

Méthode d´évitement d´obstacles

La première chose que on doit faire pour éviter les obstacles est de voir où ces obstacles peuvent se trouver. Après on doit calculer si on peut heurter ces obstacles et si nous continuerons sans changer la trajectoire qui le robot suit.

Pour voir si le robot peut heurter les obstacles on calcule la composante horizontale (selon l´axe Xr) de la mesure qui les capteurs fournissent. Si cette composante horizontale est plus petite qui la distance entre le capteur et la partie la plus externe du robot, le robot heurtera cet obstacle.

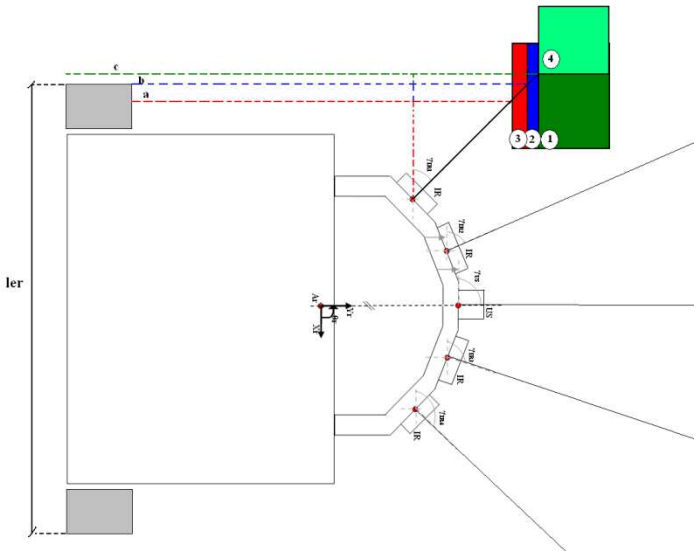

Figure 33. Robot avec des obstacles

Dans la Figure 33 on représente deux situations différentes. Première, un même objet qui change sa position (1, 2 et 3) quand le robot avance en ligne droite. Et deuxième, un objet en autre position différente à la première situation (numéro 4).

Dans la première situation on voit trois différentes positions (1, 2 et 3) et les mesures que le capteur IR1 (le premier gauche) fait du premier objet quand le robot avance en ligne droite.

- On peut noter que dans la première position de l´objet (1) la mesure que le robot fait de l´objet dit que le robot peut passer sans heurter l'objet (ligne vert, c).
- La mesure dans la deuxième position (2) détecte que l´on est à la limite pour passer (ligne bleu, b).
- Quand on est suffisamment près de l´objet (3), la composante horizontale de la mesure montre que le robot heurtera l'objet s'il ne tourne pas (ligne rouge, a).
- Dans la deuxième situation on a l'objet dans la position 4. On peut noter que le capteur donne la même mesure pour l'objet dans la position 4 que l´objet de la première situation dans la position 1. La différence est que comment on l´a vu quand le robot s´approche de l´objet dans la première situation la mesure du capteur change jusqu´a détecter qu´il doit tourner. Quand le robot s´approche de l´objet de la deuxième situation (position 4) la mesure ne va pas changer et par conséquent le robot peut passer sans changer son chemin.

A partir de cette distance limite horizontale que le robot ne doit pas dépasser on va calculer la mesure limite de chaque capteur. Si le robot respecte ces mesures limites il n'aura pas problèmes avec les objets qui se trouvent dans cet endroit. Dans ce but on a besoin les paramètres géométriques des capteurs de même que ses valeurs. On peut les voir dans les figures 34 et 35.

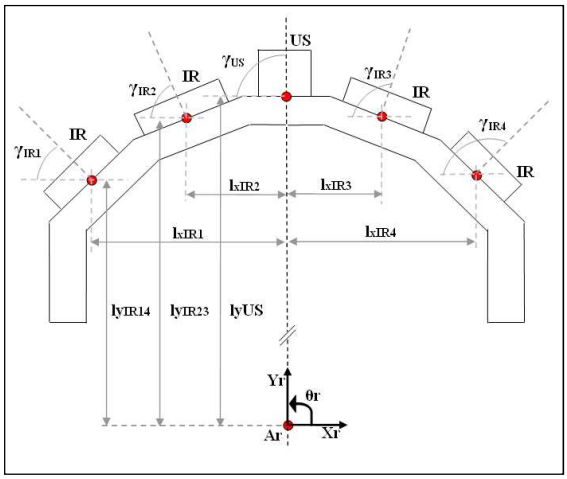

Figure 34. Paramètres Géométriques des Capteurs

Les valeurs numériques de tous ces paramètres sont listées dans le tableau suivant:

Paramètre	Valeur
γIR1	50°
γIR2	72°
γIR3	110°
γIR4	135°
γUS	90°
lyIR14	0,329 m
lyIR23	0,355 m
lyUS	0,363 m
lxIR1	-0,091 m
lxIR2	-0,044 m
lxIR3	0,042 m
lxIR4	0,090 m

Tableau 3. Tableau des Valeurs des Paramètres Géométriques

Une fois qu´on a les paramètres géométriques il reste de calculer la valeur de la mesure limite **IRxL** au moyen de simples calculs que l´on va illustrer avec le figure 35 et les équations suivantes:

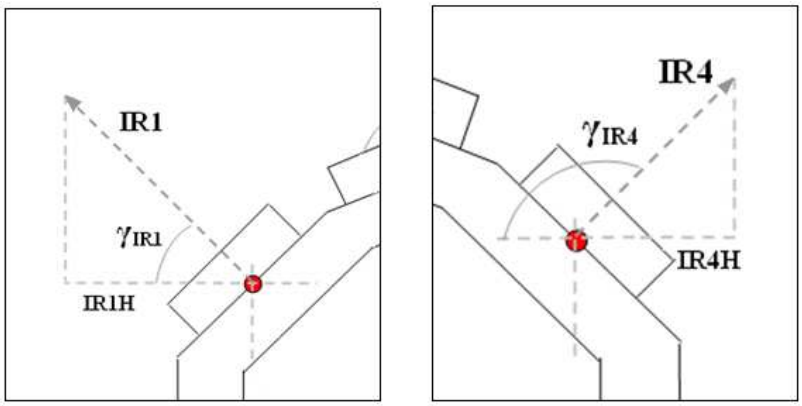

Figure 35. Composants de la mesure. Cas IR1 et IR4.

On peut voir deux différents cas.

Un cas pour les capteurs IR1 et IR2: $$IR1 = \frac{IR1H}{\cos \gamma_{IR1}}$$ (eq. 1)

Et l´autre pour les capteurs IR3 et IR4: $$IR4 = \frac{IR4H}{\cos(180 - \gamma_{IR1})}$$ (eq.2)

Pour obtenir la valeur limite on doit calculer la distance horizontale minimum qui doit avoir entre le capteur et la partie la plus externe du robot.

$$IR1HL = \frac{ler}{2} - IxIR1$$ (eq.3)

Où ler est la longueur de l'essieu du robot et il vaut 0.5 m.

Ensuite, quand on a utilisé le robot on a vu que l´on doit ajouter une valeur additionnelle à cette distance pour un bon fonctionnement du robot quand il évite les obstacles. Cette valeur est 0.1m pour les capteurs IR1 et IR2 et 0.2m pour IR3 et IR4.

Par conséquent, on a les suivants équations pour les mesures limites de chaque capteur:

$$IR1L = \frac{IR1HL}{\cos \gamma_{IR1}} = \frac{\left(\frac{ler}{2} + 0.1 - IxIR1\right)}{\cos \gamma_{IR1}}$$ (eq.4)

$$IR2L = \frac{IR2HL}{\cos \gamma_{IR2}} = \frac{\left(\frac{ler}{2} + 0.1 - IxIR2\right)}{\cos \gamma_{IR2}}$$ (eq.5)

$$IR3L = \frac{IR3HL}{\cos(180 - \gamma_{IR3})} = \frac{\left(\frac{ler}{2} + 0.2 - IxIR3\right)}{\cos(180 - \gamma_{IR3})}$$ (eq.6)

$$IR4L = \frac{IR4HL}{\cos(180 - \gamma_{IR4})} = \frac{\left(\frac{ler}{2} + 0.2 - IxIR4\right)}{\cos(180 - \gamma_{IR4})}$$ (eq.7)

Puisque le capteur ultrasonique (US), comme on peut voir dans la figure 34, n´a pas de composante horizontale la valeur limite est quelconque. On a choisi 60 cm. USL=60cm.

Modifications du code pour éviter obstacles

Le code ajouté pour éviter les obstacles se base sur une programmation en série des conditions en tenant en compte de la relation entre la mesure obtenue par les capteurs et les limites calculées auparavant. Ces relations que l´on a programmé sont très générales. Donc on peut les modifier si nécessaire avec les exceptions qu´on peut trouver.

À l´origine les conditions qui gèrent notre programme sont:

- Si la mesure du capteur US est plus petite que le limite USL, le robot doit retenir la zone où il y a plus espace pour éviter l´obstacle. Pour cela, le programme calcule la distance horizontale que les capteurs gauches et droits donnent, puis il les compare. Tout cela a été programmé dans la première version de notre programme *(Robot_HAMMI v3.1)* mais expérimentalement on a rejeté cette utilisation puisque la mesure obtenue par le capteur US est assez variable et par conséquent peu fiable. Aussi, on a vu qui on n´a pas besoin de cette mesure parce que les mesures des capteurs infrarouges suffisent pour détecter les obstacles. Si dans projets futurs on a un fonctionnement meilleur du capteur US il pourra essayer d´utiliser cette programmation qui est présenté dans l´annexe 77b).
- Si la mesure du capteur IR est plus petite que le limite IRL cela veut dire qu´il y a un obstacle et par conséquent le robot devrait tourner au côté opposé. Dans ce but on a programmé que la roue opposée au capteur qui a détecté l´objet diminue sa vitesse jusqu´à ce que la mesure du capteur soit plus grande que sa limite (le capteur ne détecte pas l´objet). Par conséquent le robot tournera du côté opposé jusqu´à ce que le robot évite l´obstacle. Ces conditions sont les mêmes pour les quatre capteurs IR. Chaque fois que cette condition est donnée un avertissement s´affiche sur l´écran où on peut voir de quel côté l´obstacle se trouve.
- Il y a autre possibilité rédigé dans notre programmation. S´il y a, en même temps, deux capteurs opposés (gauche et droit) pour lesquels les mesures sont plus petites que leur limite, alors le robot s´arrêtera parce qu´il n´y aurait pas assez d´espace pour passer.
- On peut trouver dans l´annexe b) tout le code qu´on a rédigé pour atteindre cela, de même dans cette section on a placé des commentaires pour une meilleure compréhension.

Modifications du code pour enregistrer les données

Pour une validation ultérieure du modèle créé, il faut ajouter dans le programme un code qui sert à enregistrer les données acquises pendant la réalisation des tests.

Pour plus de commodité, le logiciel que on va utiliser pour analyser les données est Matlab. Par conséquent ces données doivent être enregistrées dans un fichier externe qui peut être lu par le Matlab. Une fois qui on a le document avec les données on peut l´ouvrir et l´en analyser autant qui l´on veut.

Ce fichier va contenir toutes les informations que l´on a stockées dans la matrice « m_ donnees » de notre programme. Ce fichier est en fait une colonne (ouvrable avec Notepad par exemple) dans laquelle toutes les données sont mises les unes à la suite des autres.

Dans la figure suivante on peut voir les ressources qu'on a ajoute à l'interface de l'usager du logiciel Robot_HAMMIv2.2 pour pouvoir atteindre ce but:

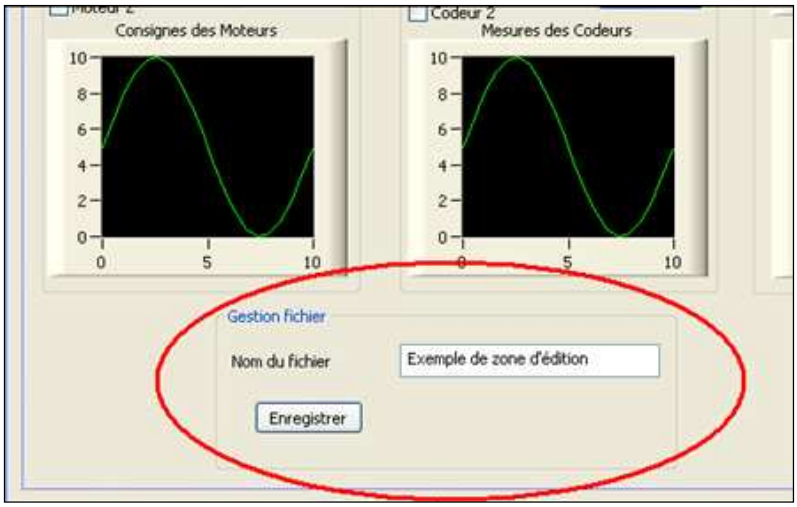

Figure 36. Partie interface de l´usager Robot_HAMMI v3.2

On peut voir que dans la partie *Gestion fichier* il y a un espace pour taper le nom du fichier que l´on donne à notre nouveau fichier. Il faut aussi d´appuyer sur le bouton *Enregistrer* pour faire l´enregistrement.

Important: Le nom du fichier ne doit pas contenir points parce ils provoquent une erreur quand le logiciel matlab tente de travailler avec ce fichier.

Si on met dans le nom du fichier un nom déjà existant il apparaît une nouvelle boite de dialogue qui prévient de l'existence d´un fichier avec le même nom et il demande si on veut replacer le fichier existant. Il s´affiche aussi si le fichier est ouvert par une autre programme.

Pour gérer tout cela, on a créé une nouvelle classe *CErreur_Nom_Fichier* avec tous ses composants, ressources et code nécessaires. On peut voir dans la figure suivante la fenêtre qui s´affiche pour gérer cette tâche.

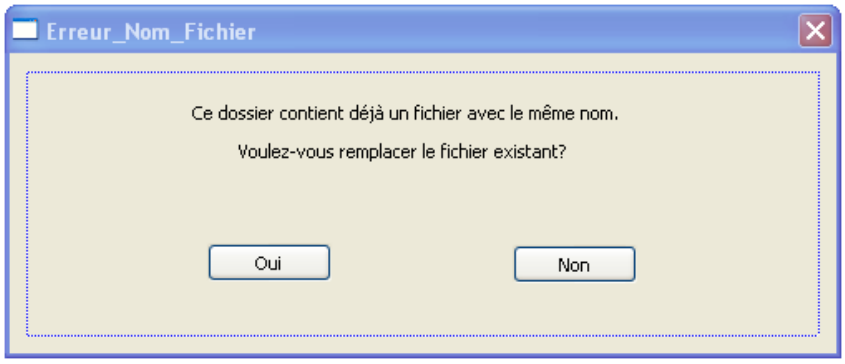

Figure 37. Boite de dialogue pour gérer les enregistrements de fichier avec le même nom.

Si on va en annexe c) on peut trouver tout le code créé pour réaliser l´enregistrement de la matrice m_donnees dans le fichier ainsi que le code pour gérer les erreurs mentionnées auparavant. Ces fichiers sont munis de commentaires pour une meilleure compréhension.

c) Création du programme pour l'analyse des données

Une fois qu'on a stocké les données dans un fichier il faut créer un programme pour l´analyse de ces données.

A partir du programme *exploi_mesu_ens* déjà créé par S.Rubrecht en Juin 2006 on a rédigé un programme Matlab adapté a nos besoins permettant d´exploiter nos données. Ce programme s´appelle ***exploi_mesu_ens_ROBOTHAMMI_v32.m***. Il dépend bien sûr du nombre de variables stockées afin de reconstruire les vecteurs de valeurs consécutives d'Omega_1, Consigne_Mot_1, etc. Ce programme a été créé car il est beaucoup plus simple d´exploiter les résultats sous Matlab que sous Visual Studio.

Une fois qu'on a reconstruit les vecteurs on a tapé le code pour avoir les graphiques dont on a besoin pour l'analyse des résultats. Ce code peut être modifié de façon facile pour s´adapter au futur besoins.

Quand on exécute le programme il nous demande le nombre de variables dans le fichier exploité. Si on laisse vide cet espace, il prend 38 comme nombre de variables par défaut. Ensuite il demande le nom du fichier à exploiter et il affiche tous les graphiques des variables stockées.

Dans l´annexe d) on peut voir le code complet avec des commentaires du programme ***exploi_mesu_ens_ROBOTHAMMI_v32.m***.

d) Validation de la méthode

Après avoir fait la programmation il faut faire la validation de ce qui a été programmé. Pour accomplir cela, on a réalisé plusieurs tests, on a stocké les données nécessaires qui servent à analyser les résultats et par conséquent à valider la méthode d´évitement d´obstacles.

Tests réalisés

Pour voir si le programme réalisé exécute le but pour lequel il a été conçu on a crée 5 test différents :

Test0: Ce premier test a été fait seulement pour voir le fonctionnement des mesures réalisées par les capteurs prennent, sans aucun objet qui perturbe les mesures.

Le robot est situé au milieu du couloir sans aucun obstacle sur son chemin. Le robot doit aller tout droit.

Figure 38. Test0 Robot_HAMMI v3.2

Test1: Premier des tests réalisés pour voir si le robot évite bien les obstacles et étudier les données reçues.

On doit dire que ici quand on parle de gauche et droite, ils sont des orientations relatives au robot dans le sens de l´avance.

Le robot est situé au coté gauche du couloir. Dans le couloir il y a deux obstacles. Le premier obstacle est à gauche, en face du robot mais seulement la partie gauche du robot se trouve avec l'obstacle si le robot ne tourne pas. Le deuxième obstacle est à droite.

Figure 39. Test1 Robot_HAMMI v3.2

Test2: Le robot est situé au coté droit du couloir. Dans le couloir il y a deux obstacles. Le premier obstacle est à droite, en face du robot mais seulement la partie droite du robot se trouve avec l'obstacle si le robot ne tourne pas. Le deuxième est à gauche.

Figure 40. Test2 Robot_HAMMI v3.2

Test3: Le robot est situé du coté droit du couloir. Dans le couloir il y a un obstacle tout en face du robot. Le robot doit l'éviter.

Figure 41. Test3 Robot_HAMMI v3.2

Test4: Le robot est situé au coté gauche du couloir. Dans le couloir il y a un obstacle tout en face du robot. Le robot doit l'éviter.

Figure 42. Test4 Robot_HAMMI v3.2

Analyse des résultats

Avant commencer l´analyse de résultats on doit tenir en compte quelque considération.

Si on va au projet qui se trouve dans la référence de bibliographie [13] on trouve que l'auteur, dans l´étude qu´il fait des capteurs et ses mesures, dit que la limite de perception des capteurs IR est entre 20 et 120 cm et pour le capteur US entre 20 et 600 cm.

Il dit aussi: « En gros, l'erreur maximale commise entre les deux données n'atteint qu'à 15 centimètres, en étant normalement au-dessous ». Cette variabilité peut être causée par la variabilité de l´angle des capteurs avec le sol causé par le mouvement du robot. On doit voir si cette variation de 15cm des mesures peut influer dans les tests réalises.

Autre chose importante est que toutes données rassemblées sont de tests à succès, c'est-à-dire, qui le robot évite bien les obstacles et il va droit dans le test0.

Pendant presque tous les tests on a filme le comportement du robot pour pouvoir revoir le comportement du robot a la même fois que les résultats et par conséquent pour un meilleur analyse.

Ensuite, on va analyser les tests qui montrent meilleur le comportement du robot.

TEST0.1_v3.2

On a réalisé 4 tests de type 0. On montre ici les résultats du TEST0.1_v3.2 comme exemple pour voir surtout le comportement des capteurs et la stabilité de ses mesures:

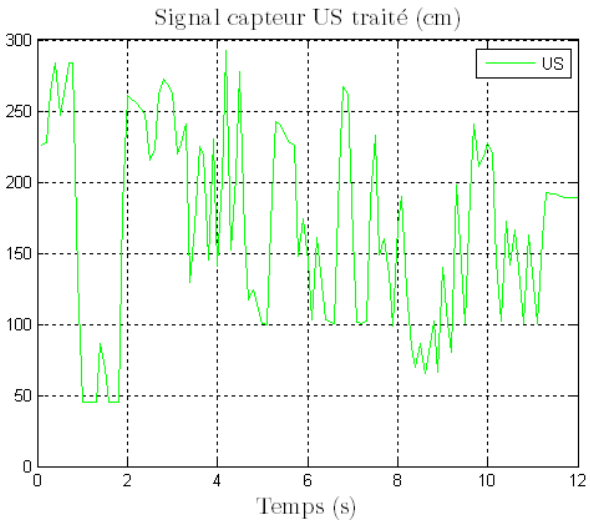

Figure 43: Signal capteur US, Test0.1_v3.2

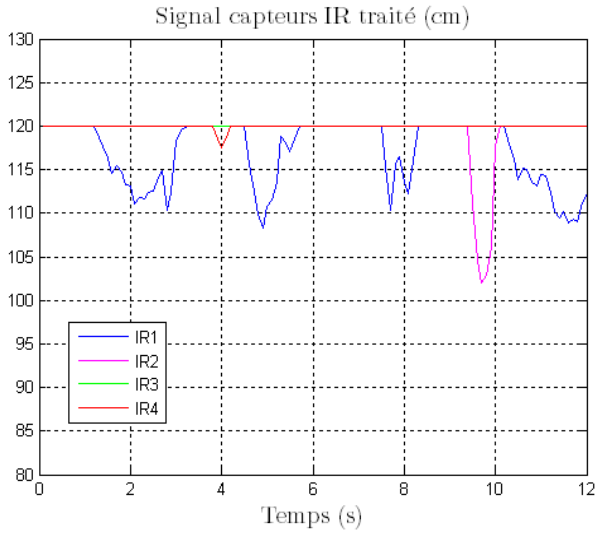

Figure 44: Signaux capteurs IR, Test0.1_v3.2

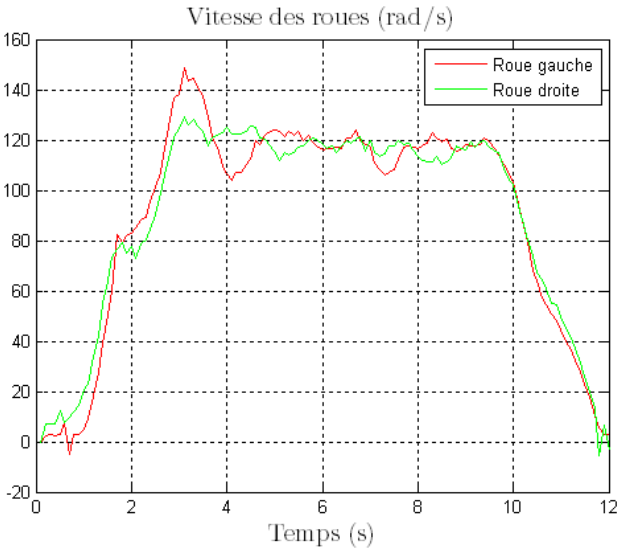

Figure 45: Vitesses des roues Test0.1_v3.2

On peut voir sur la figure 43 ce que l´on annonçait au paragraphe 6.b). La mesure obtenue par le capteur US est assez variable (dès 45cm jusqu´a 390cm) et par conséquent peu fiable. Dans un couloir sans obstacles cette mesure doit être linéairement décroissante quand on s'approche du mur à l'extrémité du couloir. On a observé la même variabilité dans tous les autres tests de tous les types.

Pour cette raison, on n´a pas tenu compte de ce capteur (US) pour la programmation de la trajectoire.

Dans la figure 44 on voit que IR3 et IR4 ne changent pas et IR1 et IR2 changent très peu. La situation idéale serait qu'IR1 et IR2 ne changent pas non plus parce qu´il n´y a aucun obstacle mais on a dit qu´il peut y avoir une variation de 15 cm qui est plus ou moins la variation observé.

Pour aller droit les vitesses des roues du robot doivent être égales. Dans la figure 45 on peut voir que les vitesses sont presque égales et par conséquent le robot ira droit.

On doit noter aussi les unités de la vitesse des roues, en rad/s, qu´on a pris du programme de l´année précédente. Cela doit être une erreur parce que la vitesse de rotation des roues atteint 140 rad/s qui correspond à 21 m/s ce qui est beaucoup. Peut-être que la mesure est en tr/min ce qui correspondrait à 2,2m/s ce qui est plus raisonnable.

Test1.6_v3.2

On a réalisé 10 tests de type 1 (7 avec vidéo), 6 de type2 (5 avec vidéo), 5 de type 3 (tous avec vidéo) et 6 de type 4 (5 avec vidéo). On montre ici les résultats du test1.6_v3.2 comme exemple qui sert pour montrer le comportement du robot de façon générale. On a choisi ce test parce que c´est un des plus clairs:

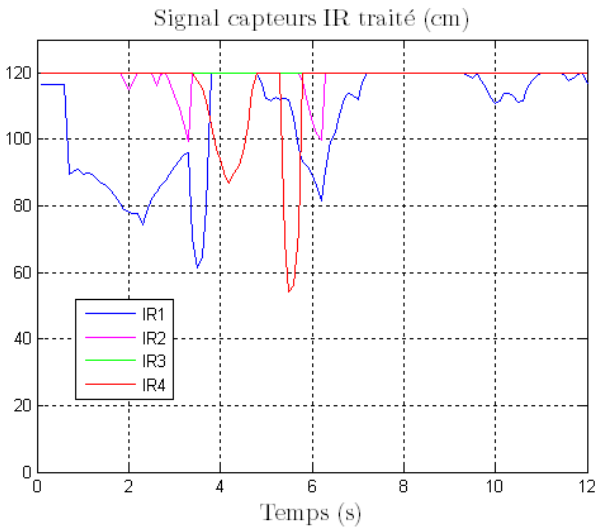

Figure 46: Signaux des capteurs IR Test1.6_v3.2

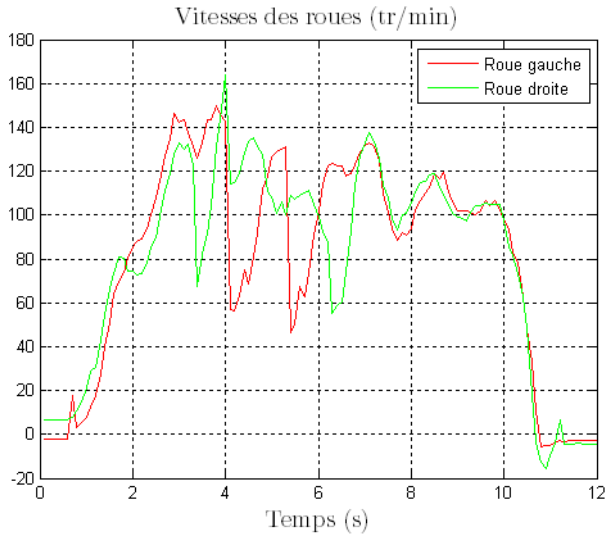

Figure 47: Vitesses des roues Test1.6_v3.2

On peut voir dans les figures 46 et 47 :

- Instant 3,5 : Le capteur IR1 détecte l´objet gauche et la vitesse de roue droite diminue pour tourner du côté droite et éviter l´obstacle.
- Instant 4,25 : Le capteur IR4 détecte le mur droit et la vitesse de roue gauche diminue pour tourner du côté gauche et ne pas heurter le mur.

- Instant 5,5 : Le capteur IR4 détecte l'objet droit et la vitesse de roue gauche diminue pour tourner du côté gauche et éviter l´obstacle.
- Instant 6,25 : Le capteur IR1 détecte le mur gauche et la vitesse de roue droite diminue pour tourner du côté droite et ne pas heurter le mur.
- Après instant 7 : Les capteurs ne détectent rien et les vitesses des roues sont les mêmes, le robot va tout droit, sans tourner.

On peut dire que la réaction du robot est bonne quand il trouve un obstacle. Nous sommes donc satisfaits du comportement du robot.

Pour plus de résultats on peut aller aux fichiers du dossier *TEST_ROBOT_HAMMI_v3.2* et exécuter ces fichiers avec le programme du matlab développé *exploi_mesu_ens_ROBOTHAMMI_v32.m*. Les test realisées sont:

Type0: TEST01_v32, TEST02_v32, TEST03_v32 et TEST04_v32

Type1: TEST11_v32, TEST12_v32, TEST13_v32, TEST14_v32, TEST15_v32, TEST16_v32, TEST17_v32, TEST18_v32, TEST19_v32 et TEST110_v32.

Type2: TEST21_v32, TEST22_v32, TEST23_v32, TEST24_v32, TEST25_v32 et TEST26_v32.

Type3: TEST31_v32, TEST32_v32, TEST33_v32, TEST34_v32 et TEST35_v32.

Type4: TEST41_v32, TEST42_v32, TEST43_v32, TEST44_v32, TEST45_v32 et TEST46_v32.

7. CONCLUSION ET TRAVAUX FUTURS

Pour terminer, on présente les objectifs atteints pendant la première et deuxième partie de ce projet et on proposera les travaux futurs qui peuvent être bon développer dans futurs projets.

Notons que dans la première partie on a analysé les projets et les travaux des années précédentes dans le but d'obtenir une connaissance du développement déjà existant et du fonctionnement du robot HAMMI.

Aussi, on a expliqué l´apprentissage et le progrès qu´on a fait quant à la programmation (architecture, ressources NI…), le langage (Visual C++) et le logiciel (Microsoft Visual Studio 2005 et Measurement Studio) qu´il faut utiliser. On a relié les renseignements de plusieurs sources et on a synthétisé ici tout ce qu'on a cru le plus intéressant pour ce projet. On a expliqué les procédés pour construire des applications et les ressources dont on dispose pour faire cela. Enfin on a montré une des applications réalisées pour renforcer l´apprentissage. Tout ce travail a été très important pour créer une base de connaissance avec laquelle on a pu travailler sur le robot et pour atteindre le but proposé pour ce projet.

Dans la deuxième partie on a fait toute l´étude directement sur le robot. On a commencé le développement de l´application capable de piloter le robot dans des endroits avec obstacles. Ce travail s´est terminé d'une manière satisfaisante avec une application dans laquelle le robot pousseur est capable de traverser un couloir avec obstacles sans rien heurter.

Autres travaux que j´ai fait d'une manière satisfaisante sont le développement des applications sur Visual Studio 2005 et Matlab pour l´analyse des données et le comportement du robot.

Pour les travaux futurs il faut continuer avec le développement de ce premier programme d'évitement d´obstacles pour le compléter. Développements futurs peuvent concerner le test du robot dans différents endroits dans lequel le robot se déplace, créer le code servant à retrouver le chemin que le robot suit une fois que l'obstacle a été évité au moyen de la localisation du robot, etc.

Dans le rapport de la référence de bibliographie [13] l´auteur propose comme travail futur l´implantation d´un filtre Kalman sur le robot. C´est une bonne idée car les mesures des capteurs seraient moins variables et par conséquent le robot pourrait mieux éviter les obstacles.

Pour finir, on peut affirmer que j´ai atteint tous les objectifs et par conséquent que le projet a été une réussite.

8. BIBLIOGRAPHIE

[9] Rapport de l'IReSP (Institut de Recherche en Santé Publique) « LE HANDICAP, NOUVEL ENJEU DE SANTE PUBLIQUE » : http://www.cnrs.fr/infoslabos/appels-offres/docs/Appel_a_projets_handicap_IReSP_2007.pdf

[10]Optimisation mécanique et programmation d'un robot mobile d'assistance aux personnes handicapées : le robot HAMMI ; Florian GENEST ; ENSAM Paris 2008.

[11]Robotique mobile pour l'assistance aux personnes handicapées : conception mécanique et implantation électronique ; rapport final de PJE ; Adrien VENGEANT ; ENSAM Paris 2007.

[12]Partie électronique du robot HAMMI ; rapport final de PJE ; Aurélien SIXTE ; ENSAM Paris 2007.

[13]Fusion de données : Application à un robot mobile. Le robot HAMMI; Javier SAÉZ CARDADOR; ENSAM Paris 2009.

[14]Visual C++ 6 ; Ivor Horton; Wrox Press 1999.

[15]Robot HAMMI : programmation du robot pour l'assistance aux personnes handicapées ; rapport final de PJE ; David NUNEZ ; ENSAM Paris 2007.

[16]Tutorial "Creating a Measurement Studio for Visual C++ 6.0 Project" dans le site de National Instruments : http://zone.ni.com/devzone/cda/tut/p/id/3177.

9. ANNEXES

a) Création d'un projet measurement studio

Cette annexe a pour but le développement d´une application avec Visual Studio et Measurement Studio de NI. Elle est obtenue à partir de la référence [16].

Cet exemple présente les instructions pour utiliser les différents classes et contrôles de Measurement Studio qui réalisent les tâches suivantes :

Dessiner et établir avec contrôles slides les amplitudes de deux signaux sinusoïdaux.

Ajouter un bouton qui montre l'amplitude du second slide sur un indicateur numérique. Il actualise aussi le graphique avec le plot du second slide.

Programmer les changements de couleur, style de ligne, largeur de ligne des graphiques en concordance avec son amplitude.

Table de Matières:

1. Créer le Dialogue

2. Modifier les Contrôles

3. Ajouter Variables pour les Contrôles

4. Ajouter Fonctions Membres pour les Contrôles

5. Ajouter le Code au Projet

6. Ajouter propriétés Avancées au Projet

CREATION DU DIALOGUE

Compléter les différentes étapes pour créer le dialogue et lui ajouter les contrôles :

Ouvrez Visual Studio 2005 et cliquez sur ***Fichier→Nouveau→ Projet* .** Choisir dans le menu de Visual C++ l'option **Measurement Studio** et puis cliquez sur **NI MFC Application**.

Tapez ***Projet*** sur la casse **Nom**.

Choisissez un répertoire pour enregistrer le projet.

Cliquez ***OK***.

Cliquez ***Next*** en acceptant toutes les options par défaut sauf ***User Interface Features*** où l'on doit désélectionner l'option ***About Box***. Continuez en cliquant ***Next*** jusqu' à il ne soit pas possible cliquer sur Next. Pressez finalement ***Finish***.

Désélectionnez tous les composants sauf ***Common, Professional Analysis*** et ***User interface*** et puis cliquez ***Finish***.

Dans le panneau ***Affichage de ressources***, développez le dialogue et double cliquez sur ***IDD_PROJET_DIALOG***. Dans la fenêtre, supprimez le texte ***TODO : Placez ici les contrôles de boîtes de dialogues***. Effacez aussi les boutons ***OK*** et ***Annuler***.

Traînez vers la ressource depuis l'éditeur de boîte à outils un ***CWSlide Control*** (National Instruments) et un ***CWGraph Control*** (National Instruments). Placez les contrôles comme le montre

la Figure A-1. Pour redimensionner les contrôles sur la ressource du dialogue, sélectionnez le contrôle et étirez le coin.

NOTE: Si les contrôles n'apparaissent pas, cliquez le bouton droit de la souris sur le cadre de la fenêtre et sélectionnez ***Controls***.

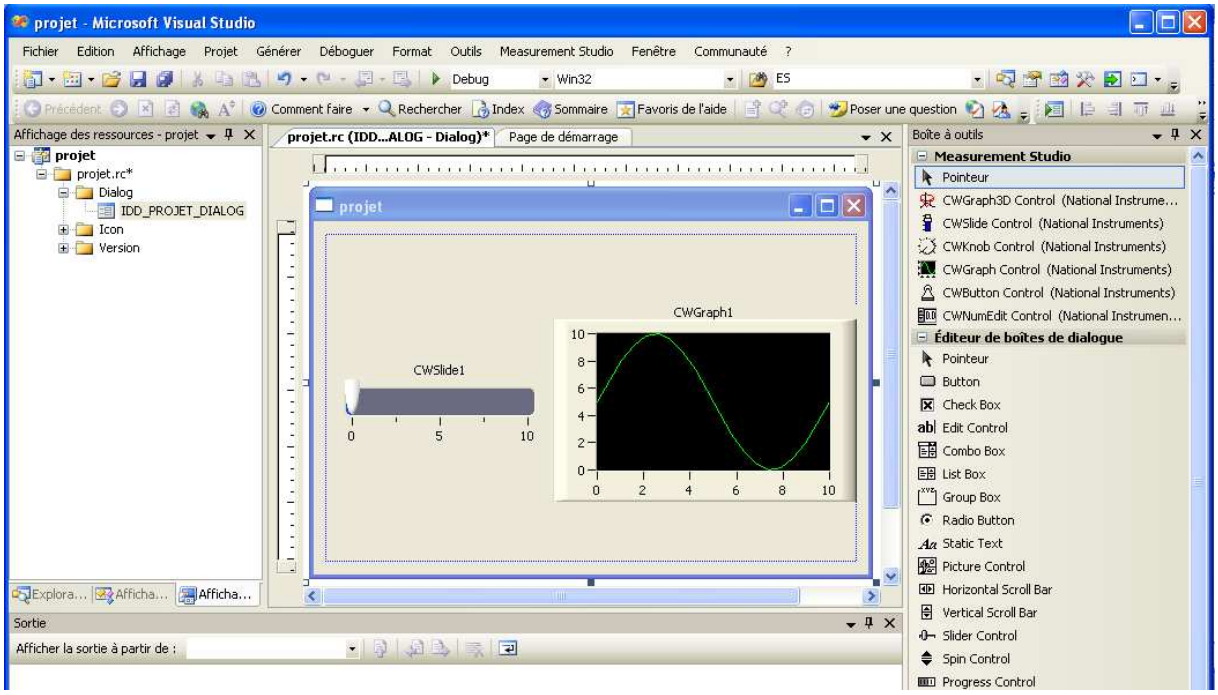

Figure A - 1: Projet Dialogue initial

MODIFICATION DES CONTRÔLES

On peut établir les propriétés des contrôles telles que les noms, l'échelle de valeurs des contrôles numériques et les configurations des axes. Réaliser les étapes suivantes pour modifier les contrôles:

1. Cliquez le bouton droit de la souris sur un contrôle quelconque et choisissez *Propriétés* pour afficher la page de propriétés (c'est le dernier onglet de la fenêtre).
2. Pour chaque contrôle, écrivez dans la case *Caption* du slide contrôle, *Dash Dot* et supprimez le titre du contrôle graphique.
3. Pour le contrôle graphique, cliquez sur le dernier onglet de la fenêtre de *Propriétés* et puis cliquez *Plots*. Changez le type de ligne à *Dash Dot*.
4. Pour établir l'échelle de l'axe Y sur le graphique, cliquez sur *Axes* dans la même page. Sélectionnez *Y-Axes1* et puis désélectionnez l'option *Auto scale*. Tapez –10 sur la case *Minimum*.
5. Pour le contrôle du slide, répétez les mêmes étages que pour le contrôle graphique pour cliquer sur le dernier onglet. Cliquez sur *Numeric* et dans la section *Scale*, établissez le *Minimum* à 0 et le *Maximum* à 5. Quand les propriétés du contrôle sont affichées, choisissez le *Style* et sélectionnez le style Slide *3D Vertical Pointer.*

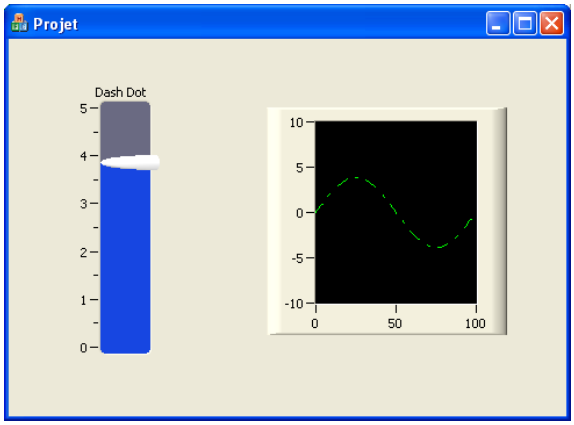

Figure A - 2: Modification 1

INSERTION DES VARIABLES POUR LES CONTRÔLES

On peut ajouter variables pour les contrôles pour établir ses propriétés et appeler ses fonctions lors l'exécution du programme. Réaliser les étapes suivantes pour ajouter les noms aux variables de contrôle:

1. Cliquez le bouton droit de la souris sur le graphique et puis cliquez ***Ajouter une variable.***
2. Notez que le type de variable membre est déjà défini comme NI::CNiGraph. Tapez ***m_graph*** dans la case ***Nom*** *de la variable* et cliquez ***Terminer***.
3. Répétez les étapes 1 et 2. Faites de l'attention avec **l'ID du contrôle**. Il doit être IDC_CWSLIDE1, puis changez le nom de la variable à m_slide1.
4. Cliquez ***Terminer***.

NOTE: On peut voir les modifications faites dans le fichier d'en tête ProjetDlg.h.

INSERTION DES FONCTIONS MEMBRES AUX CONTRÔLE

On ajoute les fonctions membres aux contrôles pour répondre aux événements que le contrôle génère:

1. Cliquez le bouton droit de la souris sur la fenêtre et puis cliquez Propriétés.
2. Cliquez sur l'onglet événements de contrôle et développez **IDC_CWSLIDE1**. Cliquez sur la case ***PointerValueChanged*** et puis ***<Ajouter>*** PointerValueChangedCwslide1.
3. Ouvrez les fichiers *ProjetDlg.cpp* et vérifiez que la fonction membre a été ajoutée.

INSERTION DU CODE AU PROJET

Réaliser les étapes suivantes pour ajouter le code au projet:

1. Ouvrez l'affichage de classes et cherchez la classe **CProjetDlg** dans le dossier *Types derivés*. Cliquez le bouton droit de la souris sur cette classe et sélectionnez ***Ajouter→Ajouter une variable***. Tapez **CNiReal64Vector** sur la case du *Type de variable* et *m_wave* sur ***Nom de la variable***. Sélectionnez ***Private*** dans la case ***Accès***. Vous allez utiliser cette variable pour enregistrer les données du signal. CNiReal64Vector est un type de variable qui stocke les données dans un vecteur de doubles.
2. Cliquez *Terminer*.
3. Pour ajouter le code à la fonction membre, dans l'affichage de classes, cliquez le bouton gauche de la souris sur l'item ***CProjetDlg*** et puis faites du double click sur **OnInitDialog**.
4. Ouvrez le fichier ***ProjetDlg.cpp***, dans la fonction **OnInitDialog**, ajoutez le code suivant en gras.

```
BOOL CProjetDlg::()
{
   CDialog::OnInitDialog();

   // Définir l'icône de cette boîte de dialogue.
   // L'infrastructure effectue cela
   // automatiquement lorsque la fenêtre
   // principale de l'application n'est pas
   // une boîte de dialogue
   SetIcon(m_hIcon, TRUE);   // Définir une grande icône
   SetIcon(m_hIcon, FALSE);  // Définir une petite icône
   // Générer le vecteur de 100 données du sinus
   // Amplitude = 1
   CNiMath::SineWave(m_wave,100,1);
   // Plot du premier signal
   m_graph.Plots.Item(1).PlotY(m_wave);
   // On établit la valeur du slide contrôle
   // avec l'amplitude adéquate
   m_slide1.Value=1;
return TRUE;  // retourne TRUE, sauf si vous avez défini le
         // focus sur un contrôle
}
```

5. Cliquez le bouton gauche de la souris sur le contrôle ***Slide*** et puis cliquez ***Propriétés***. Sélectionnez ***Événements de contrôle*** et puis cliquez ***PointerValueChanged***. Développez la liste déroulante et sélectionnez ***<Ajouter>PointerValueChangedCliquezCwslide1***.
6. Ajouter le code qui est écrit ci dessous en gras. Ce code change le signal sinusoïdal quand la valeur du slide correspondant change.

```
void CProjetDlg::PointerValueChangedCwslide1(long Pointer, VARIANT* Value)
{
   //Création d'une copie locale du signal
   CNiReal64Vector temp(m_wave);
   //Modifier amplitude du sinus selon le contrôle slide
   temp.Scale(CNiVariant(Value));
   //Mettre à jour le premier plot
   m_graph.Plots.Item(1).PlotY(temp);
}
```

7. Cliquez ***Générer→Générer Projet*** pour compiler et vérifier que le projet n'a pas d'erreurs. Si la compilation a réussi, on peut exécuter le programme. Cliquez ***Déboguer→Exécuter sans débogage***. Vérifier que l'amplitude du graphique change en fonction de la valeur du slide **Dash Dot**.

INSERTION DES PROPRIÉTÉS AVANCÉES AU PROJET

Les étapes suivantes montrent comment on peut programmer les changements de couleur et du style des lignes des graphiques. La propriété ***CNiPlot, LineStyle***, est un type de données énuméré. On peut chercher la configuration en cliquant ***Measurement Studio→NI Measurement Studio Help*** et utilisez l'index pour trouver l'information sur CNiPlot.

On va ajouter des propriétés avancées à l'exemple avec lequel on travaille:

1. Ajoutez un contrôle **CWSlide**, un bouton **CWButton**, **CWNumEdit**, et un contrôle Visual C++ Group Box à la ressource de la boîte de dialogue. Replacez les contrôles selon la figure A-3.

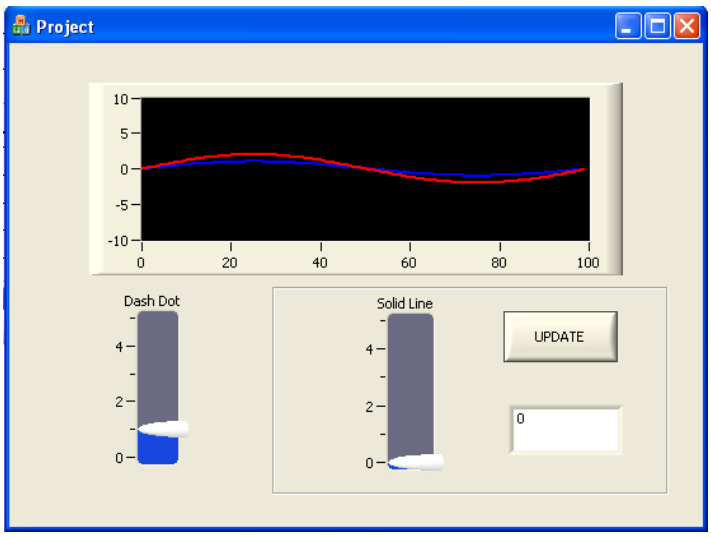

Figure A - 3: Contrôles supplémentaires

NOTE : Le bouton Update est fondé sur le style de bouton de MeasurementStudio3D mais on peut aussi utiliser le style de bouton de Visual C++. Si on utilise le style de bouton Visual C++, on doit créer un lien Click pour l'événement *Click*.

2. Ajoutez les titres des contrôles supplémentaires si nécessaires.
3. Ajoutez un *Static* ***Text*** "Solid Line Plot Value".
4. Utilisez le menu Plots de la page de propriétés du graphique. Cliquez ***Add*** dans la section ***Plots*** pour ajouter un plot supplémentaire au graphique. Changez la couleur de la ligne en rouge. Cliquez ***OK***.
5. Ajoutez des variables membres pour les contrôles ***Numedit*** et ***slide***. Consultez l'aide sur ***Ajouter*** *des variables de contrôle* pour plus d'informations. Cliquez le bouton droit de la souris sur chaque contrôle et puis cliquez ***Ajouter une variable***. Automatiquement, la nouvelle variable sera associée à l'ID du contrôle. Utilisez les noms **m_numedit** pour le contrôle ***Numedit*** et **m_slide2** pour le second contrôle ***Slide***.
6. Ajoutez une fonction membre pour linker le message **Click** envoyé quand on presse le bouton ***Update***. Consultez l'aide sur ***Ajouter des fonctions membres de contrôle*** pour plus d'informations. Cliquez le bouton droit de la souris sur le bouton ***Update*** et puis cliquez sur ***Propriétés***. Cliquez l'onglet de ***Evénements de contrôle*** et ajouter le message **ClickCwboolean1**.
7. Ajoutez le code suivant à la fin de la fonction **CProjetDlg::OnInitDialog** juste avant la dernière instruction.

```
// Création d'un vecteur de travail initialisé à la valeur
// de la variable membre m_wave. Double le signal
// sinusoïdal. Affiche le signal.

CNiReal64Vector temp(m_wave);  // Tampon
```

```
temp.Scale(2);              // Double amplitude
m_graph.Plots.Item(2).PlotY(temp);   // Affiche
```

8. Ajoutez le code suivant écrit en gras à la fonction

CProjetDlg ::ClickCwboolean1.

```
void CProjetDlg::ClickCwboolean1()
{
  // On met à jour le second graphique et son indicateur
  // numérique de l'amplitude

  m_numedit.Value=m_slide2.Value;
  CNiReal64Vector temp(m_wave);    // Réserve tampon
  temp.Scale(m_slide2.Value);      // Modifie amplitude
  m_graph.Plots.Item(2).PlotY(temp);
}
```

9. Pour changer les couleurs des graphes, remplacez le code de la fonction `PointerValueChangedCwSlide1` par le code écrit en gras ci dessous.

```
void CProjetDlg::PointerValueChangedCwslide1(long Pointer, VARIANT* Value)
{
   // Lecture slide et stockage
   double dValue=CNiVariant(Value);
   // Copie la première courbe dans plot
   CNiPlot plot=m_graph.Plots.Item(1);
   // On crée une copie locale du sinus dans temp
   CNiReal64Vector temp(m_wave);
   // Modifie amplitude en fonction du slide
   temp.Scale(dValue);

   if(dValue<2)
   {
      plot.LineColor=CNiColor(0,0,255); // bleu
      plot.LineStyle=CNiPlot::LineSolid;
   }

   else if(dValue<4)
   {
      plot.LineColor=CNiColor(0,255,0); // vert
      plot.LineStyle=CNiPlot::LineDash;
   }

   else
   {
      plot.LineColor=CNiColor(255,0,0); // rouge
      plot.LineStyle=CNiPlot::LineDot;
   }
```

```
    // On dessine l'onde sinusoïdale nouvelle
    plot.PlotY(temp);
}
```

10. Cliquez ***Générer→Générer Projet*** pour compiler et vérifier que le projet n'a pas d'erreurs. Si la compilation a réussi, on peut exécuter le programme. Cliquez ***Déboguer→Exécuter sans débogage***. Vérifiez que le graphique change selon la valeur du slide *Dash Dot*. Vérifiez aussi que si on change le contrôle ***Solid Line*** et si on clique le bouton ***Update***, le contrôle ***Solid Line Plot Value*** montre la valeur correcte et met à jour le graphe correspondent.

    La Figure A-4 montre un exemple de ce qui devrait apparaître si l'application fonctionne.

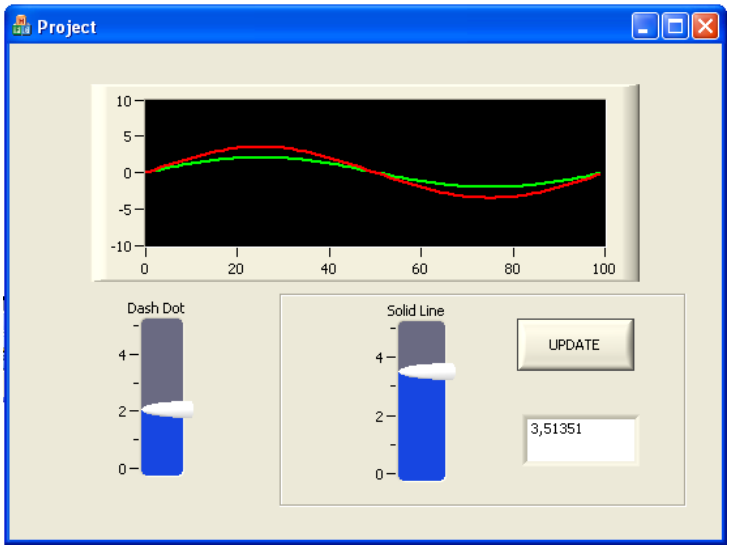

Figure A - 4: Présentation finale

b) Code ajouté pour eviter les obstacles

Dans cette annexe on présentera les modifications au code du logiciel *Robot_HAMMI v2.2* pour obtenir la version actuelle, décrite dans le paragraphe 6.b).

Les modifications accomplies pour éviter obstacles son principalement faites dans le fichier *Robot_HAMMIView.cpp*. Les changements du code les plus importants de ce fichier sont dans le gestionnaire du temporisateur appelé *OnTimer*:

Commençons avec la programmation fait dans ***Robot_HAMMI v3.1*** et qui comme on a dit a été finalement rejeté (paragraphe 6.b)). On présente ce programme au cas où peut être utile dans projets futurs où la mesure du capteur US serait moins variable:

```
//-----------------------------------------------------
// Code servant a eviter obstacles
// Ajouté par Khalid Bashir Rodriguez en Fevrier 2010 ///////////////

// Definition des variables à utiliser
```

```
// Valeur des paramétres géométriques des capteurs
double thetaIR1=50*pi/180 ; // en rad
double thetaIR2=72*pi/180 ;
double thetaIR3=110*pi/180 ;
double thetaIR4=135*pi/180 ;
double lxIR1= 0.091 ; // en m
double lxIR2= 0.044 ;
double lxIR3= 0.042 ;
double lxIR4= 0.090 ;


// Vérification d'obstacles avec les données récemment obtenues et programmation pour les éviter

double IR1L=(((ler/2+0.1)-lxIR1)/cos(thetaIR1))*100 ; //en cm
double IR2L=(((ler/2+0.1)-lxIR2)/cos(thetaIR2))*100;
double IR3L=(((ler/2+0.2)-lxIR3)/cos(pi-thetaIR3))*100;
double IR4L=(((ler/2+0.2)-lxIR4)/cos(pi-thetaIR4))*100;
double USL=60;

if (TR_Capteurs[4] < USL)  // On detecte un obstacle
{
// On va chercher où il y a beacoup d'espace pour passer et on va tourner par là
///////// ON NE PEUT PAS COMPARER TR_Capteurs[2] ET TR_Capteurs[1] PARCE QUE NE SONT PAS SYMETRIQUES ET LES ANGLES NON PLUS. MEME ENTRE TR_Capteurs[0] ET TR_Capteurs[3]
///////// ON PEUT CALCULER AVEC SES PROJECTIONS EN Xr DES ET APRES LES COMPARER

///////// Projection en Xr des mesures des capteurs: IR1H, IR2H, IR3H, IR4H

double IR1H=TR_Capteurs[0]*cos(thetaIR1); // en cm
double IR2H=TR_Capteurs[1]*cos(thetaIR2);
double IR3H=TR_Capteurs[2]*cos(pi-thetaIR3);
double IR4H=TR_Capteurs[3]*cos(pi-thetaIR4);

        //D'abord on vérifie les capteur infrarrouges avant
        if (IR3H < IR2H) // il y a beacoup d'espace pour le côté gauche
        {
                v_moteur[0]=v_moteur[1]*0.5;  // On diminue la vitesse de roue1(gauche) pour tourner du coté avec beacoup d´espace(gauche)
                AfxMessageBox("IR3H<IR2H");
        }
        else
        {
                if (IR2H < IR3H) // il y a beacoup d'espace pour le côté droite
                {
                v_moteur[1]=v_moteur[0]*0.5;  // On disminue la vitesse de roue2(droite) pour tourner au coté avec beacoup d´espace(droite)
                AfxMessageBox("IR2H<IR3H");
                }
                else
                {
                        if (IR2H == IR3H)
                        {
                        //Maintenant on vérifie les capteur infrarrouges arrière
                        if (IR4H < IR1H)
                        {
                        v_moteur[0]=v_moteur[1]*0.5;
                        AfxMessageBox("IR4H<IR1H");
```

```
                        }
                        else
                        {
                                if (IR1H < IR4H)
                                {
                                v_moteur[1]=v_moteur[0]*0.5;
                                AfxMessageBox("IR1H<IR4H");
                                }
                                else
                                {
                                        v_moteur[1]=v_moteur[0]*0.5;  //On diminue la vitesse du roue2 pour
tourner du coté droit (choix aléatoire)
                                        AfxMessageBox("ALEATOIRE");
                                }
                        }
                        }
                }
        }

}
else

        // Obstacle pour les côtés (gauche ou droite)
{
if (((TR_Capteurs[0] < IR1L) || (TR_Capteurs[1] < IR2L)) && ((TR_Capteurs[2] < IR3L) || (TR_Capteurs[3] < IR4L)))

        {
        OnBnClickedArreter();
        AfxMessageBox("OBJET EN INTERROMPANT LE TRAJECTOIRE.PAS DE PLACE SUFFISANTE POUR PASSER");
        }
        else
        {
                while ((TR_Capteurs[2] < IR3L) || (TR_Capteurs[3] < IR4L)) // Obstacle pour les côté droite
                {
                v_moteur[0]=v_moteur[1]*0.5;  // On diminue la vitesse de roue1(gauche) pour tourner du coté avec
beacoup d´espace(gauche)
                CCommandes_moteurs_Code envoi_consignes;       // Crée un objet de la classe CCd_Mot_1_Code
                envoi_consignes.WriteData(v_moteur);      // Envoi des consignes des vitesses aux moteurs
                Sleep(600);                   //Attend que la nouvelle vitesse soit acquise physiquement
                AfxMessageBox("OBSTACLE POUR LE COTE DROITE");

                }

                while ((TR_Capteurs[0] < IR1L) || (TR_Capteurs[1] < IR2L)) // Obstacle pour les côté gauche
                {
                v_moteur[1]=v_moteur[0]*0.5;  // On diminue la vitesse de roue2(droite) pour tourner au coté avec
beacoup d´espace(droite)
                CCommandes_moteurs_Code envoi_consignes;       // Crée un objet de la classe CCd_Mot_1_Code
                envoi_consignes.WriteData(v_moteur);      // Envoi des consignes des vitesses aux moteurs
                Sleep(600);                   //Attend que la nouvelle vitesse soit acquise physiquement

                AfxMessageBox("OBSTACLE POUR LE COTE GAUCHE");

                }

        }
```

```
}

CCommandes_moteurs_Code envoi_consignes;       // Crée un objet de la classe CCd_Mot_1_Code
envoi_consignes.WriteData(v_moteur);       // Envoi des consignes des vitesses aux moteurs

//-------------------------------------------------------------------------------
```

Maintenant la programmation fait dans ***Robot_HAMMI v3.2*** et qui est la utilisé finalement:

```
//-----------------------------------------------------
// Code servant a eviter obstacles
// Ajouté par Khalid Bashir Rodriguez en Fevrier 2010 ///////////////

// Definition des variables à utiliser
// Valeur des paramétres géométriques des capteurs
double thetaIR1=50*pi/180 ; // en rad
double thetaIR2=72*pi/180 ;
double thetaIR3=110*pi/180 ;
double thetaIR4=135*pi/180 ;
double lxIR1= 0.091 ; // en m
double lxIR2= 0.044 ;
double lxIR3= 0.042 ;
double lxIR4= 0.090 ;


// Vérification d'obstacles avec les données récemment obtenues et programmation pour les éviter

double IR1L=(((ler/2+0.1)-lxIR1)/cos(thetaIR1))*100 ; //en cm
double IR2L=(((ler/2+0.1)-lxIR2)/cos(thetaIR2))*100;
double IR3L=(((ler/2+0.2)-lxIR3)/cos(pi-thetaIR3))*100;
double IR4L=(((ler/2+0.2)-lxIR4)/cos(pi-thetaIR4))*100;
double USL=60;


if (((TR_Capteurs[0] < IR1L) || (TR_Capteurs[1] < IR2L)) && ((TR_Capteurs[2] < IR3L) || (TR_Capteurs[3] < IR4L)))
        {
                OnBnClickedArreter();
                AfxMessageBox("OBJET EN INTERROMPANT LE TRAJECTOIRE.PAS DE PLACE SUFFISANTE POUR PASSER");
        }
        else
        {
                while ((TR_Capteurs[2] < IR3L) || (TR_Capteurs[3] < IR4L)) // Obstacle pour le côté droit
                {
                v_moteur[0]=v_moteur[1]*0.5;  // On diminue la vitesse de roue1(gauche) pour tourner au coté avec beacoup d´espace(gauche)
                CCommandes_moteurs_Code envoi_consignes;          // Crée un objet de la classe CCd_Mot_1_Code
                envoi_consignes.WriteData(v_moteur);// Envoi des consignes des vitesses aux moteurs
                Sleep(600);                //Attend que la nouvelle vitesse soit acquise physiquement
```

```
                AfxMessageBox("OBSTACLE POUR LE COTE DROITE");

                }
                while ((TR_Capteurs[0] < IR1L) || (TR_Capteurs[1] < IR2L)) // Obstacle pour les côté gauche
                {
                v_moteur[1]=v_moteur[0]*0.5;  // On diminue la vitesse de roue2(droite) pour tourner au coté avec beacoup d´espace(droite)
                CCommandes_moteurs_Code envoi_consignes;          // Crée un objet de la classe CCd_Mot_1_Code
                envoi_consignes.WriteData(v_moteur);// Envoi des consignes des vitesses aux moteurs
                Sleep(600);                //Attend que la nouvelle vitesse soit acquise physiquement


                AfxMessageBox("OBSTACLE POUR LE COTE GAUCHE");

                }

        }


CCommandes_moteurs_Code envoi_consignes;          // Crée un objet de la classe CCd_Mot_1_Code
envoi_consignes.WriteData(v_moteur);// Envoi des consignes des vitesses aux moteurs


//----------------------------------------------------------------------------
```

c) Code ajouté pour enregister les données

Cette annexe a par but celui de montrer et d'expliquer les principaux éléments du code pour l´enregistrement de la matrice *m_donnees* dans un fichier externe à Visual Studio2005 et aussi ce code nécessaire pour gérer l´erreur qui s´affiche quand l'utilisateur veut enregistrer des données dans un fichier sous le nom d'un autre fichier déjà existant ou bien si le fichier est ouvert par un autre programme. Ce logiciel est exposé dans le chapitre 6.b).

Voici le code qui gère l´enregistrement des données dans un fichier. Il se trouve dans le fichier *Robot_HAMMIView.cpp* de notre programme

```
// Inclusion du fichier d'en tête de la classe qui gère
// la boîte de dialogue qui montre le message d'erreur
// malgré à l'existence d'un fichier avec le même nom.
#include "Erreur_Nom_Fichier.h"
#include <fstream>      // #include nécessaire pour gérer les fichiers (pour enregister la matice m_donnees)
#include <iostream>     // #include nécessaire pour gérer les E/S
using namespace std;

// Fonction qui gére l'enregistrement des données dans un fichier
// Ajouté par Khalid Bashir Rodriguez en Fevrier 2010 ///////////////

void CRobot_HAMMIView::OnBnClickedEnregistrer()
```

```
{
	// Actualisation du nom du fichier d'enregistrement
	UpdateData(TRUE);
	UpdateData(FALSE);

	// Déclaration du flux de données (stream)
	fstream fichier;

	// Déclaration d'un stream de lecture pour tester
	// si le fichier est déjà existant
	ifstream test(m_nom_fichier);

	// Boucle qui est exécutée si le fichier existe ou s'il est déjà ouvert
	if(test.is_open())
	{
		remplacer=FALSE;	// Au début, on ne veut pas remplacer le fichier

		// Appel à la boîte de dialogue qui affiche le message
		// de l'existence d'un fichier avec le même nom
		CErreur_Nom_Fichier dlg_Fichier;	// Création d'un objet de cette classe
		dlg_Fichier.DoModal();				// Appel à la boîte de dialogue modale

		// Traite le cas où l'utilisateur veut remplacer le fichier précédant
		if(remplacer==TRUE)
		{
			// Ouverture du fichier pour l'enregistrement de données
			fichier.open(m_nom_fichier,ios::out);
			if(!fichier)
				// Message d'erreur
				cout<<"Erreur à l'ouverture du fichier"<<endl;
			else
			{
				// Boucle qui enregistre les données dans
				// le fichier désiré par l'utilisateur
				for(int j=1;j<=nb_echantillons;j++)
				{
					for(int i=0;i<=nb_variables;i++)
						// Enregistrement de la donnée
						fichier<<m_donnees(i,j)<<endl; // fin du for sur i
				}	// fin du for sur j
				fichier.close();  // Fermeture du fichier
			}	// fin du else
			fichier.close();
		}	// fin du if(remplacer==TRUE)
		test.close();
	}	// fin du if(test.is_open())

	// Le fichier n'existe pas dans le dossier de travail
	else
	{
		// Ouverture du fichier pour l'enregistrement de données
```

```
		fichier.open(m_nom_fichier,ios::out);
		if(!fichier)
			// Message d'erreur
				cout<<"Erreur à l'ouverture du fichier"<<endl;
		else
		{
			// Boucle qui fait l'enregistrement des données
			// dans le fichier désirée par l'utilisateur
			for(int j=1;j<=nb_echantillons;j++)
			{
				for(int i=0;i<=nb_variables;i++)
					// Enregistrement de la donnée
					fichier<<m_donnees(i,j)<<endl; // fin du for sur i

			}	// fin du for sur j
			// Fermeture du fichier
			fichier.close();
		}	// fin du else(!fichier)
		test.close();
	}	// fin du else(test.is_open())
}
```

Voici le code nécessaire pour gérer l´erreur qui s´affiche quand l'utilisateur veut enregistrer des données dans un fichier sous le nom d'un autre fichier déjà existant ou bien si le fichier est ouvert par un autre programme.

Erreur_Nom_Fichier.h

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//          FICHIER MODIFIE PAR
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//        Khalid BASHIR RODRÍGUEZ
//                  Fevrier 2010
//                  E.N.S.A.M. Paris
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
////////////////////////////////////////////////////////////////////////////////////////////////////
//////              Fichier d'en tête qui gère la création et la manipulation
//////              de la classe CErreur_Nom_Fichier. Cette classe est
//////              responsable de la boite de dialogue modale qui s'affiche
//////              quand l'utilisateur veut enregistrer des données dans un
//////              fichier sous le nom d'un autre fichier déjà existant ou
//////              bien si le fichier est ouvert par un autre programme.
//////              Tout ce fichier a été généré automatiquement par le
//////              logiciel sauf l'instruction commentée à suite.
////////////////////////////////////////////////////////////////////////////////////////////////////
#pragma once

// On doit inclure le fichier "Resource.h" puisque ce fichier d'en tête
// doit reconnaître la boîte de dialogue associée à cette classe.
#include "Resource.h"
```

```
// Boîte de dialogue CErreur_Nom_Fichier

class CErreur_Nom_Fichier : public CDialog
{
	DECLARE_DYNAMIC(CErreur_Nom_Fichier)

public:
	CErreur_Nom_Fichier(CWnd* pParent = NULL);   // constructeur standard
	virtual ~CErreur_Nom_Fichier();

// Données de boîte de dialogue
	enum { IDD = IDD_DIALOG_NOM_FICHIER };

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // Prise en charge de DDX/DDV

	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnBnClickedErreurNomFichierOui();
	afx_msg void OnBnClickedErreurNomFichierNon();
};
```

Erreur_Nom_Fichier.cpp

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//           FICHIER MODIFIE PAR
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//          Khalid BASHIR RODRÍGUEZ
//                    Fevrier 2010
//                    E.N.S.A.M. Paris
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////              Fichier .cpp qui gère l'implémentation et la manipulation
//////              de la classe CErreur_Nom_Fichier. Cette classe est la
//////              responsable de la boite de dialogue modale qui s'affiche
//////              quand l'utilisateur veut enregistrer des données dans un
//////              fichier sous le nom d'un autre fichier déjà existant ou
//////              bien si le fichier est ouvert par un autre programme.
////////////////////////////////////////////////////////////////////////////////////////////////////////////


// Erreur_Nom_Fichier.cpp : fichier d'implémentation

#include "stdafx.h"
#include "Robot_HAMMI.h"
#include "Erreur_Nom_Fichier.h"

// Inclusion du fichier qui définit toutes les variables globales
// Dans ce cas, DRAPEAU n'est pas défini donc est le deuxième
// bloc de ce fichier qui est exécuté.
#include "Variables_globales.h"
```

```
// Boîte de dialogue CErreur_Nom_Fichier

IMPLEMENT_DYNAMIC(CErreur_Nom_Fichier, CDialog)

CErreur_Nom_Fichier::CErreur_Nom_Fichier(CWnd* pParent /*=NULL*/)
	: CDialog(CErreur_Nom_Fichier::IDD, pParent)
{

}

CErreur_Nom_Fichier::~CErreur_Nom_Fichier()
{
}

void CErreur_Nom_Fichier::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

// Déclaration des gestionnaires d'événements des boutons de la boîte de dialogue.
// Cette partie est ajoutée automatiquement par le logiciel quand on ajoute les
// événements de contrôle.
BEGIN_MESSAGE_MAP(CErreur_Nom_Fichier, CDialog)
	ON_BN_CLICKED(IDC_ERREUR_NOM_FICHIER_OUI,
&CErreur_Nom_Fichier::OnBnClickedErreurNomFichierOui)
	ON_BN_CLICKED(IDC_ERREUR_NOM_FICHIER_NON,
&CErreur_Nom_Fichier::OnBnClickedErreurNomFichierNon)
END_MESSAGE_MAP()

//-------------------------------------------------------------------
// Gestionnaires de messages de CErreur_Nom_Fichier

// Fonction qui retourne le contrôle au programme principal en sachant
// que l'utilisateur veut remplacer le fichier déjà existant

void CErreur_Nom_Fichier::OnBnClickedErreurNomFichierOui()
{
	remplacer=TRUE;		// L'utilisateur veut remplacer le fichier
	EndDialog(1);		// Fermeture de la boîte de dialogue
}

//-------------------------------------------------------------------

// Fonction qui retourne le contrôle au programme principal en sachant
// que l'utilisateur ne veut pas remplacer le fichier déjà existant

void CErreur_Nom_Fichier::OnBnClickedErreurNomFichierNon()
{
	remplacer=FALSE;		// L'utilisateur ne veut pas remplacer le fichier
	EndDialog(1);		// Fermeture de la boîte de dialogue
}
//-------------------------------------------------------------------
```

d) Programme crée pour le analyses des donnees

Le but de cette annexe est de présenter le code du fichier de *Matlab* appelé ***exploi_mesu_ens_ROBOTHAMMI_v32.m***. Ce fichier sert à vérifier la validité du logiciel *Robot_HAMMI v3.2* en représentant des graphiques pour l´analyse de résultats.

Les données introduites dans ce fichier sortent d'un fichier qui est créé par le logiciel Robot_Hammi_v3.2 si on appuie sur le bouton *enregistrer* de l´interface de l´usager. Voici le code :

```
%Explotation des mesures sur Robot_Hammi_V3.2
%Fichier créé par Khalid Bashir -Fevrier 2010- a partir
%du fichier exploi_mesu_ens créé par S. Rubrecht-Juin 2006-

%Ce fichier sert à exploiter les données après une expérience.



%%%%%ATTENTION. LES FICHIERS QUE CE PROGRAMME VA TRAITER NE DOIVENT PAS
%%%%%CONTENIR POINT DANS SE NOM. UN FICHIER TEST1.1_V3 DONNE UN ERREUR.
%%%%%LE NOM DU FICHIER DOIT ETRE PAR EXEMPLE TEST11_V3

%=========================================
clc; clear all;close all;

Tech=0.1;   %période d'échantillonnage
rr=0.15;    %rayon de la roue du robot
ler=0.50;   %longueur d'essieu du robot

disp('Fichiers disponibles')
dir *.

disp=('Donner le nombre de variables dans le fichier exploité :  equivalent a nb_variables (variable globale du prog C++');
disp=('0 pour la valeur par défaut (38)');
n=input('Donner le nombre de variables dans le fichier exploité : n = ' )
if n==0
  nbrevariablesmesurees=38;
else
nbrevariablesmesurees=n;
end



fich=input('choisir un nom de fichier sans extension:','s');
eval(['load ',fich]);%lit le fichier de donéées brut à exploiter
fichier=eval(fich);%récupère le nom du fichier
donnees=fichier;
nbrevariablesmesurees=38; % equivalent a nb_variables (variable globale du
                    %prog C++)

nb_ech = length(donnees)/nbrevariablesmesurees;%calcul du nombre
                    %d'échantillons de l'expérience

conversion1= 1.025/81;%facteur de conversion: tensionmot= conversion*omegamot
conversion2= 1.035/81;
```

```
reduction=36;%coefficient de reduction du motoreducteur

%Attention, dans le for ci dessous à bien avoir le même ordre que dans le
%programme C++
for kk=1:nb_ech,
  temps(1,kk) = donnees(nbrevariablesmesurees*(kk-1)+1);
  consi_omega1(1,kk) = donnees(nbrevariablesmesurees*(kk-1)+2);%rad/s
  consi_omega2(1,kk) = donnees(nbrevariablesmesurees*(kk-1)+3);%rad/s


  mesu_omega1(1,kk) = donnees(nbrevariablesmesurees*(kk-1)+4);%rad/s
  mesu_omega2(1,kk) = donnees(nbrevariablesmesurees*(kk-1)+5);%rad/s
  xr(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+6);
  yr(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+7);
  thetar(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+8);
  theta11(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+9);
  theta12(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+10);


  xd(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+11);
  yd(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+12);

  ex(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+13);
  ey(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+14);

  IR0(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+15);
  IR1(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+16);
  IR2(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+17);
  IR3(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+18);
  US(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+19);

  TR_Capteurs0(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+20);
  TR_Capteurs1(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+21);
  TR_Capteurs2(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+22);
  TR_Capteurs3(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+23);
  TR_Capteurs4(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+24);

  ES_Capteurs0(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+25);
  ES_Capteurs1(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+26);
  ES_Capteurs2(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+27);
  ES_Capteurs3(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+28);
  ES_Capteurs4(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+29);

  ER_Capteurs0(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+30);
  ER_Capteurs1(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+31);
  ER_Capteurs2(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+32);
  ER_Capteurs3(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+33);
  ER_Capteurs4(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+34);

  xmur1(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+35);
  xmur2(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+36);
  ymur1(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+37);
  ymur2(1,kk)=donnees(nbrevariablesmesurees*(kk-1)+38);

end
```

```
figure(1)
hold on
set(gcf,'Color','w');
BOX ON
grid on
plot(temps, mesu_omega1,'-r');
plot(temps, mesu_omega2,'-g');
legend('Roue gauche','Roue droite')
title('Vitesse des roues (rad/s)','fontsize',14);
xlabel('Temps (s)','fontsize',14);
hold off

figure(2)
hold on
set(gcf,'Color','w');
BOX ON
grid on
plot(temps,TR_Capteurs0,'-b');
plot(temps,TR_Capteurs1,'-m');
plot(temps,TR_Capteurs2,'-g');
plot(temps,TR_Capteurs3,'-r');
legend('IR1','IR2','IR3','IR4')
title('Signal capteurs IR traité (cm)','fontsize',14);
xlabel('Temps (s)','fontsize',14);
YLIM([0 130]);
hold off

figure(3)
hold on
set(gcf,'Color','w');
BOX ON
grid on
plot(temps,TR_Capteurs0,'-g');
plot(temps,TR_Capteurs1,'-r');
legend('IR1','IR2')
title('Signal capteurs IR1 et IR2 traité (cm)','fontsize',14);
xlabel('Temps (s)','fontsize',14);
YLIM([0 130]);
hold off

figure(4)
hold on
set(gcf,'Color','w');
BOX ON
grid on
plot(temps,TR_Capteurs2,'-g');
plot(temps,TR_Capteurs3,'-r');
legend('IR3','IR4')
title('Signal capteurs IR3 et IR4 traité (cm)','fontsize',14);
xlabel('Temps (s)','fontsize',14);
YLIM([0 130]);
hold off

figure(10)
hold on
```

```
set(gcf,'Color','w');
BOX ON
grid on
plot(temps,TR_Capteurs4,'-g');
legend('US')
title('Signal capteur US traité (cm)','fontsize',14);
xlabel('Temps (s)','fontsize',14);
hold off
```